NORMA
BRASILEÑA

ABNT NBR 15606-4

Primera edición
13.04.2010

Válida a partir de
13.05.2010

Televisión digital terrestre — Codificación de datos y especificaciones de transmisión para radiodifusión digital
Parte 4: Ginga-J — Ambiente para la ejecución de aplicaciones procedurales

ICS 33.080; 33.160.01

ISBN 978-85-07-02128-5

Número de referencia
ABNT NBR 15606-4:2010
99 páginas

ABNT
Av.Treze de Maio, 13 - 28º andar
20031-901 - Rio de Janeiro - RJ
Tel.: + 55 21 3974-2300
Fax: + 55 21 39742346
abnt@abnt.org.br
www.abnt.org.br

Impresso en Brasil

Índice

Página

Prólogo ix
Introducción x
1 Alcance 1
2 Referencias normativas 1
3 Términos y definiciones 2
4 Abreviaturas 3
5 Arquitectura del *middleware* Ginga 3
5.1 Visión general de la arquitectura Ginga 3
5.2 Arquitectura Ginga-J 4
5.2.1 Contexto 4
5.2.2 Arquitectura 5
6 Formato del contenido 5
7 Modelo de aplicación Ginga-J 5
7.1 Modelo de aplicación 5
7.1.1 Ciclo de vida 5
7.1.2 Inicio de aplicaciones 7
7.1.3 Finalización de aplica 7
7.1.4 Soporte a múltiples aplicaciones 8
7.1.5 Compartimiento de recursos entre aplicaciones 8
7.1.6 Controlando aplicaciones 8
7.1.7 Comunicación entre aplicaciones 8
7.1.8 Propiedades del ambiente 9
7.1.9 Códigos de control de aplicación 9
7.2 Almacenamiento y *caching* de aplicaciones 11
7.2.1 Modelos de almacenamiento 11
7.2.2 Cuestiones de almacenamiento 11
7.2.3 *Caching* proactivo 12
7.3 Transmisión de aplicaciones 12
7.3.1 Reglas de señalización 12
7.3.2 Empaquetamiento de aplicaciones 12
7.3.3 Autenticación de aplicación 12
7.3.4 Señalizando la misma aplicación en diversos servicios 12
7.3.5 *Download* de aplicaciones a través del canal interactivo 13
8 Plataforma Ginga-J 13
8.1 Plataforma Java 13
8.2 Consideraciones básicas de la plataforma 13
8.2.1 Ambiente de ejecución 13
8.2.2 Jerarquía de paquetes y clases 14
8.2.3 Notificación de eventos 14
8.2.4 Codificación de texto 14
8.2.5 Ciclo de vida de las aplicaciones 14

8.3 Infraestructura común 15
8.4 Presentación gráfica y tratamiento de eventos 16
8.4.1 LWUIT, *LightWeight user interface toolkit* 16
8.4.2 Interfaz gráfica de usuario 16
8.4.3 Tratamiento de los planos de la plataforma 17
8.4.4 Tratamiento de eventos del usuario 23
8.5 Información y selección de servicios 23
8.5.1 Consideraciones generales 23
8.5.2 Integración con API independiente de protocolo 24
8.6 Presentación y ejecución de medias 24
8.7 Acceso a datos 24
8.7.1 Consideraciones generales 24
8.7.2 Acceso a archivos 24
8.7.3 Protocolo de transporte por radiodifusión 24
8.7.4 Almacenamiento persistente 25
8.7.5 Acceso a las propiedades del sistema 25
8.7.6 Soporte a IP sobre canal de interactividad 25
8.7.7 Filtración de sección MPEG-2 25
8.8 Gestión de aplicaciones 26
8.9 Sintonización 26
8.10 Puente NCL 26
8.11 Propiedades de la plataforma 26
8.11.1 Propiedades de sistema 26
8.11.2 Propiedades de usuario 27
8.12 Canal de interactividad 27
8.13 Lista de paquetes mínimos del Ginga-J 27
8.13.1 Paquetes de la plataforma Java 27
8.13.2 Paquetes de la especificación JavaTV 1.1 y JMF 1.0 28
8.13.3 Paquetes de la especificación JAVADTV 1.3 29
8.13.4 Paquetes de la especificación JSSE 1.0.1 30
8.13.5 Paquetes de la especificación JCE 1.0.1 31
8.13.6 Paquetes de la especificación SATSA 1.0.1 31
8.13.7 Paquetes específicos Ginga-J 31
Anexo A (normativo) **Especificación JavaDTV 1.3** 32
A.1 Consideraciones generales 32
A.2 API Java DTV 32
A.2.1 Paquete com.sun.dtv.broadcast 32
A.2.2 Paquete com.sun.dtv.smartcard 33
A.2.3 Paquete com.sun.dtv.lwuit.events 33
A.2.4 Paquete com.sun.dtv.filtering 34
A.2.5 Paquete com.sun.dtv.ui.event 35
A.2.6 Paquete com.sun.dtv.lwuit.plaf 36
A.2.7 Paquete com.sun.dtv.media.timeline 37

A.2.8 Paquete com.sun.dtv.media.language **37**
A.2.9 Paquete com.sun.dtv.application **38**
A.2.10 Paquete com.sun.dtv.media.audio **38**
A.2.11 Paquete com.sun.dtv.test **39**
A.2.12 Paquete com.sun.dtv.tuner **39**
A.2.13 Paquete com.sun.dtv.lwuit.layouts **40**
A.2.14 Paquete com.sun.dtv.broadcast.event **41**
A.2.15 Paquete com.sun.dtv.lwuit.list **41**
A.2.16 Paquete com.sun.dtv.ui **42**
A.2.17 Paquete com.sun.dtv.media.control **43**
A.2.18 Paquete com.sun.dtv.media.dripfeed **44**
A.2.19 Paquete com.sun.dtv.security **44**
A.2.20 Paquete com.sun.dtv.lwuit.painter **45**
A.2.21 Paquete com.sun.dtv.locator **45**
A.2.22 Paquete com.sun.dtv.resources **45**
A.2.23 Paquete com.sun.dtv.net **46**
A.2.24 Paquete com.sun.dtv.media.text **46**
A.2.25 Paquete com.sun.dtv.media.format **47**
A.2.26 Paquete com.sun.dtv.platform **48**
A.2.27 Paquete com.sun.dtv.io **48**
A.2.28 Paquete com.sun.dtv.lwuit.animations **48**
A.2.29 Paquete com.sun.dtv.service **49**
A.2.30 Paquete com.sun.dtv.media **49**
A.2.31 Paquete com.sun.dtv.transport **50**
A.2.32 Paquete com.sun.dtv.lwuit.util **50**
A.2.33 Paquete com.sun.dtv.lwuit **51**
A.2.34 Paquete com.sun.dtv.lwuit.geom **52**
Anexo B (normativo) **Especificación de la API de informaciones de servicio dependiente de protocolo** **54**
B.1 Consideraciones generales **54**
B.2 API información de servicio dependiente de protocolo **54**
B.2.1 Paquete br.org.sbtvd.net **54**
B.2.1.1 Clase *SBTVDLocator* **54**
B.2.1.2 Clase *SBTVDNetworkBoundLocator* **56**
B.2.2 Paquete *br.org.sbtvd.si* **56**
B.2.2.1 Interfaz *DescriptorTag* **56**
B.2.2.2 Interfaz *PMTElementaryStream* **62**
B.2.2.3 Interfaz *PMTService* **62**
B.2.2.4 Interfaz *PMTStreamType* **63**
B.2.2.5 Interfaz *SIBouquet* **64**
B.2.2.6 Interfaz *SIBroadcaster* **65**
B.2.2.7 Interfaz *SIEvent* **65**
B.2.2.8 Interfaz *SIInformation* **67**

B.2.2.9 Interfaz *SIIterator* 68
B.2.2.10 Interfaz SIMonitoringListener 68
B.2.2.11 Interfaz *SIMonitoringType* 69
B.2.2.12 Interfaz *SINetwork* 69
B.2.2.13 Interfaz *SIRetrievalListener* 70
B.2.2.14 Interfaz *SIRunningStatus* 70
B.2.2.15 Interfaz *SIService* 71
B.2.2.16 Interfaz *SIServiceType* 72
B.2.2.17 Interfaz *SITime* 74
B.2.2.18 Interfaz *SITransportStream* 74
B.2.2.19 Interfaz *SITransportStreamBAT* 75
B.2.2.20 Interfaz *SITransportStreamNIT* 75
B.2.2.21 Clase *Descriptor* 75
B.2.2.22 Clase *SIDatabase* 75
B.2.2.23 Clase *SIExEventInformation* 79
B.2.2.24 Clase *SILackOfResourcesEvent* 79
B.2.2.25 Clase SIMonitoringEvent 79
B.2.2.26 Clase *SINotInCacheEvent* 80
B.2.2.27 Clase *SIObjectNotInTableEvent* 81
B.2.2.28 Clase *SIRequest* 81
B.2.2.29 Clase *SIRequestCancelledEvent* 81
B.2.2.30 Clase *SIRetrievalEvent* 81
B.2.2.31 Clase *SISuccessfulRetrieveEvent* 82
B.2.2.32 *Clase SITableNotFoundEvent* 82
B.2.2.33 Clase *SITableUpdatedEvent* 82
B.2.2.34 Clase *SIUtil* 82
B.2.2.35 Clase *SIException*() 83
B.2.2.36 Clase *SIIllegalArgumentException()* 83
B.2.2.37 Clase *SIInvalidPeriodException* 83
Anexo C (normativo) **Especificación API de extensión para sintonía – Paquete *br.org.sbtvd.net.tuning*** 84
C.1 Clase *ChannelManager* 84
C.2 Clase *Channel* 85
Anexo D (normativo)**Especificación API de puente NCL** 86
D.1 Consideraciones generales 86
D.2 API de puente NCL 86
D.2.1 Paquete br.org.sbtvd.bridge 86
D.2.1.1 Clase *NCLPlayer* 86
D.2.1.2 Clase *NCLPlayerEvent* 88
D.2.1.3 Interfaz *NCLPlayerEventListener* 89
D.2.1.4 Clase *NCLGingaSettingsNode* 90
D.2.1.5 Clase *NCLEdit* 90
D.2.2 Paquete br.org.sbtvd.bridge.ncl 94

D.2.2.1 Clase *NodeManager* 94
D.2.2.2 Clase *NCLEvent* 94
D.2.2.3 Interfaz *NCLEventListener* 96
Anexo E (normativo) **Especificación API de soporte a planos gráficos – Paquete *br.org.sbtvd.ui*** 97
E.1 Clase *ColorCoding* 97
E.2 Clase *StillPicture* 97
E.3 Clase *SwitchArea* 98
Bibliografía 99

Figuras

Figura 1 – Arquitectura en alto nivel del *middleware* Ginga 4
Figura 2 – Contexto del Ginga-J 4
Figura 3 – Arquitectura Ginga-J y ambiente de ejecución 5
Figura 4 – Diagrama con estados del ciclo de vida de un Xlet 6
Figura 5 – Estructura de capas para la presentación de servicios 17
Figura 6 – Ejemplo de composición de los plano de exhibición de contenido 22

Tablas

Tabla 1 – Propiedades del ambiente Ginga-J 9
Tabla 2 – Códigos de control de aplicaciones Ginga-J 9
Tabla 3 – Detalle de funcionalidades para el plano de texto y gráficos 18
Tabla 4 – Detalle de funcionalidades para el plano de video e imágenes 19
Tabla 5 – Detalle de funcionalidades para el plano de imágenes estáticas 20
Tabla 6 – Detalle de funcionalidades para el plano de video 21
Tabla 7 – Mapeado de las funciones de la ABNT NBR 15604:2007 en los eventos definidos por la JAVADTV 1.3:2009 23
Tabla 8 – Propiedades de sistema 27
Tabla A.1 – Clases del paquete com.sun.dtv.broadcast 32
Tabla A.2 – Clases del paquete com.sun.dtv.smartcard 33
Tabla A.3 – Clases del paquete com.sun.dtv.lwuit.events 33
Tabla A.4 – Clases del paquete com.sun.dtv.filtering 34
Tabla A.5 – Clases del paquete com.sun.ui.event 36
Tabla A.6 – Clases del paquete com.sun.lwui.plaf 37
Tabla A.7 – Clases del paquete com.sun.dtv.media.timeline 37
Tabla A.8 – Clases del paquete com.sun.dtv.media.language 38
Tabla A.9 – Clases del paquete com.sun.dtv.application 38
Tabla A.10 – Clases del paquete com.sun.dtv.media.auto 39
Tabla A.11 – Clases del paquete com.sun.dtv.test 39
Tabla A.12 – Clases del paquete com.sun.dtv.tuner 39
Tabla A.13 – Clases del paquete com.sun.dtv.lwuit.layouts 40

Tabla A.14 – Clases del paquete com.sun.dtv.broadcast.event 41
Tabla A.15 – Clases del paquete com.sun.dtv.lwuit.list 41
Tabla A.16 – Clases del paquete com.sun.dtv.ui 42
Tabla A.17 – Clases del paquete com.sun.dtv.media.control 44
Tabla A.18 – Clases del paquete com.sun.dtv.media.dripfeed 44
Tabla A.19 – Clases del paquete com.sun.dtv.security 44
Tabla A.20 – Clases del paquete com.sun.dtv.lwuit.painter 45
Tabla A.21 – Clases del paquete com.sun.dtv.locator 45
Tabla A.22 – Clases del paquete com.sun.dtv.resources 46
Tabla A.23 – Clases del paquete com.sun.dtv.net 46
Tabla A.24 – Clases del paquete com.sun.dtv.media.text 47
Tabla A.25 – Clases del paquete com.sun.dtv.media.format 47
Tabla A.26 – Clases del paquete com.sun.dtv.platform 48
Tabla A.27 – Clases del paquete com.sun.dtv.io 48
Tabla A.28 – Clases del paquete com.sun.dtv.lwuit.animations 49
Tabla A.29 – Clases del paquete com.sun.dtv.service 49
Tabla A.30 – Clases del paquete com.sun.dtv.media 49
Tabla A.31 – Clases del paquete com.sun.dtv.transport 50
Tabla A.32 – Clases del paquete com.sun.dtv.lwuit.util 51
Tabla A.33 – Clases del paquete com.sun.dtv.lwuit 51
Tabla A.34 – Clases del paquete com.sun.dtv.lwuit.geom 53

Prólogo

La Associação Brasileira de Normas Técnicas (ABNT) es el Foro Nacional de Normalización. Las Normas Brasileñas, cuyo contenido es responsabilidad de los Comités Brasileños (ABNT/CB), de los Organismos de Normalización Sectorial (ABNT/ONS) y de las Comisiones de Estudios Especiales (ABNT/CEE), son elaboradas por Comisiones de Estudio (CE), formadas por representantes de sus sectores implicados de los que forman parte: productores, consumidores y neutrales (universidades, laboratorios y otros).

Los Documentos Técnicos ABNT se elaboran de acuerdo con las reglas estabelecidas en la Parte 2 de las Directivas ABNT.

La Associação Brasileira de Normas Técnicas (ABNT) llama la atención sobre la posibilidad de que algunos de los elementos de este documento pueden ser objeto de derechos de patente. La ABNT no debe ser considerada responsable por la identificación de cualesquiera derechos de patente.

La ABNT NBR 15606-4 fue elaborada en la Comisión de Estudio Especial de Televisión Digital (ABNT/CEE-85). Su primer Proyecto circuló en Consulta Nacional conforme Edicto nº 09, de 06.09.2007 a 05.11.2007, con el número de Proyecto 00:001.85-006/4. Su 2º Proyecto circuló en Consulta Nacional conforme Edicto nº 05, de 19.05.2009 a 17.07.2009, con el número de 2º Proyecto 00:001.85-006/4. Su 3º Proyecto circuló en Consulta Nacional conforme Edicto nº 03, de 05.03.2010 a 05.04.2010, con el número de 3º Proyecto 85:000.00-006/4.

En caso de que surja cualquier duda con relación a la interpretación de la versión en español siempre deben prevalecer las prescripciones de la versión en portugués.

Esta Norma está basada en los trabajos del Fórum del Sistema Brasileño de Televisión Digital Terrestre, conforme establecido en el Decreto Presidencial nº 5.820, de 29.06.2006.

Esta versión en español es equivalente a la ABNT NBR 15606-4:2010, de 13.04.2010.

Esta versión en español fue publicada en 24.06.2010.

El Alcance de esta Norma Brasileña en inglés es el siguiente:

Scope

This part of ABNT NBR 15606 specifies the requirements for the procedural part of the middleware for the Brazilian digital terrestrial televison system (SBTVD).

Introducción

La definición Ginga-J se compone de un conjunto de Interfaces de Programación de Aplicativos (API – Application Programming Interfaz) proyectadas para suplir todas las funcionalidades necesarias para la implementación de aplicativos para televisión digital, desde la manipulación de datos multimedia hasta protocolos de acceso.

La especificación Ginga se aplica a los receptores para sistemas de transmisión terrestre de televisión (over-the-air). Ginga se destina a cubrir una serie completa de implementaciones incluyendo los receptores-decodificadores integrados (IRD), aparatos de televisión integrados, computadoras multimedia y clusters locales de aparatos conectados vía redes domésticas (Home Area Networks, HAN).

Esta parte de la ABNT NBR 15606 se destina a los desarrolladores de receptores compatibles con el sistema brasileño de televisión digital terrestre (SBTVD) y a los desarrolladores de aplicativos que utilizan la funcionalidad y API Ginga.

Esta parte de la ABNT NBR 15606 tiene como objetivo garantizar la interoperabilidad de los aplicativos Ginga y diferentes implementaciones Ginga.

Esta parte de la ABNT NBR 15606 es congruente con especificaciones internacionales, conforme a lo detallado en el Anexo A.

Televisión digital terrestre — Codificación de datos y especificaciones de transmisión para radiodifusión digital
Parte 4: Ginga-J — Ambiente para la ejecución de aplicaciones procedurales

1 Alcance

Esta parte de la ABNT NBR 15606 especifica los requisitos para la parte procedural del *middleware* para el sistema brasileño de televisión digital terrestre (SBTVD).

2 Referencias normativas

Los documentos relacionados a seguir son indispensables a la aplicación de este documento. Para referencias fechadas, se aplican apenas las ediciones citadas. Para referencias no fechadas, se aplican las ediciones más recientes de dicho documento (incluyendo enmiendas).

ABNT NBR 15601:2007, *Televisión digital terrestre – Sistema de transmisión*

ABNT NBR 15603:2007 (todas las partes), *Televisión digital terrestre – Multiplexación y servicios de información (SI)*

ABNT NBR 15604:2007, *Televisión digital terrestre – Receptores*

ABNT NBR 15606-1, *Televisión digital terrestre – Codificación de datos y especificaciones de transmisión para radiodifusión digital - Parte 1: Codificación de datos*

ABNT NBR 15606-2:2007, *Televisión digital terrestre – Codificación de datos y especificaciones de transmisión para radiodifusión digital Parte 2: Ginga-NCL para receptores fijos y móviles – Lenguaje de aplicación XML para codificación de aplicaciones*

ABNT NBR 15606-3:2007, *Televisión digital terrestre – Codificación de datos y especificaciones de transmisión para radiodifusión digital - Parte 3: Especificación de transmisión de datos*

ISO 639-2, *Codes for the representation of names of languages – Part 2: alpha-3 code.*

ISO/IEC 8859-1:1998, *Information technology - 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet Nº 1*

ISO/IEC 13818-1, *Information technology – Generic coding of moving pictures and associated audio information: Systems*

ARIB STD-B10:2008, *Service information for digital broadcasting system*

ARIB STD-B23:2006, *Application execution engine platform for digital broadcasting*

ARIB STD-B31:2007, *Transmission Coding Standard*

CDC 1.1:2008, *Connected Device Configuration 1.1 (JSR218), disponible en http://jcp.org/en/jsr/detail?id=218*

FP 1.1:2008, *Foundation Profile 1.1 (JSR 219), disponible en http://jcp.org/en/jsr/detail?id=219*

JAR:2009, Sun Microsystems. JAR File Specification. 2009, disponible en *http://java.sun.com/j2se/1.4.2/docs/guide/jar/jar.html*

LWUIT 1.1:2008 , *LightWeight User Interface Toolkit, Sun Microsystems*

JAVADTV1.3:2009, *Java DTV Specification, Sun Microsystems*

JAVATV1.1:2008, *Java TV Specification 1.1 (JSR 927), Sun Microsystems disponible en: http://jcp.org/en/jsr/detail?id=927*

JCE:1.0.1:2006, *Sun Microsystem. Security (JCE – Java Cryptography Extension) Optional Package Specification v1.0.1, disponible en: http://jcp.org/en/jsr/detail?id=219*

JSSE 1.0.1:2006, *Sun Microsystem. Security (JSSE – Java Secure Socket Extension) Optional Package Specification v1.0.1, disponible en: http://jcp.org/en/jsr/detail?id=219*

JVM:1997, *Java(TM) Virtual Machine Specification, The (2nd Edition), T Lindholm, F Yellin – 1997 – Addison-Wesley*

PBP 1.1:2008, *Personal Basis Profile 1.1 (JSR 217), disponible en http://jcp.org/en/jsr/detail?id=217*

SATSA:1.0.1:2007, *Sun Microsystem, Security and Trust Services API for J2ME (JSR 177), disponible en http://jcp.org/en/jsr/detail?id=177*

3 Términos y definiciones

Para los efectos de esta parte de la ABNT NBR 15606, se aplican los siguientes términos y definiciones.

3.1
bytecode
forma intermediaria de código interpretada por la JVM

3.2
contexto de servicio
ambiente en el cual el servicio se presenta en el receptor digital

3.3
máquina virtual Java
Java virtual machine
JVM
proceso que carga y ejecuta los aplicativos Java

3.4
servicio
conjunto de informaciones, que contiene audio, video y/o datos, para presentación en un receptor digital

NOTA Normalmente el servicio es referenciado por los televidentes como “canal de televisión”.

3.5
zapper
aplicación residente, típicamente desarrollada por el fabricante del receptor la cual el usuario puede activar en cualquier momento

NOTA O zapper se puede usar para seleccionar servicios y aplicaciones para posterior ejecución.

4 Abreviaturas

Para los efectos de esta parte de la ABNT NBR 15606, se aplican las siguientes abreviaturas.

API	*Application Programming Interface*
CA	*Conditional Access*
CDC	*Connected Device Configuration*
CSS	*Cascading Style Sheets*
ECMA	*European Computer Manufacturers Association*
EDT	*Event*
EPG	*Electronic Program Guide*
HAN	*Home Area Network*
IRD	*Integrated Receiver Decoder*
JPEG	*Joint Photographic Expert Group*
MIDP	*Móbile Information Devide Profile*
MPEG	*Moving Picture Expert Group*
NCL	*Nested Context Language*
PBP	*Personal Basis Profile*
PNG	*Portable Network Graphics*
SBTVD	*Sistema Brasileño de Televisión Digital Terrestre*
TOT	*Time Offset Table*
TS	*Transport Stream*
UTF	*Unicode Transformation Format*
XHTML	*eXtensible Hypertext Markup Language*

5 Arquitectura del *middleware* Ginga

5.1 Visión general de la arquitectura Ginga

El universo de las aplicaciones para televisión digital puede ser particionado en dos conjuntos: el de las aplicaciones declarativas y el de las aplicaciones procedurales. Una aplicación declarativa es aquella en que su entidad "inicial" es del tipo "contenido declarativo". Análogamente, una aplicación procedural es aquella en que su entidad "inicial" es del tipo "contenido procedural".

Un contenido declarativo se debe basar en un lenguaje declarativo, es decir, en un lenguaje que enfatiza la descripción declarativa del problema, en lugar de su descomposición en una implementación algorítmica. Un contenido procedural se debe basar en un lenguaje no declarativo. Lenguajes no declarativos pueden seguir diferentes paradigmas. Tenemos así, los lenguajes basados en módulos, orientadas a objetos etc. La literatura sobre televisión digital, sin embargo, utiliza el término procedural para representar todos los lenguajes que no son declarativos. En una programación procedural, la computadora debe obligatoriamente ser informada sobre cada paso a ser ejecutado. Se puede afirmar que, en lenguajes procedurales, el programador posee un mayor poder de expresión en su código, siendo capaz de establecer todo el flujo de control y ejecución de su programa - como existen más recursos disponibles el grado de complejidad es mayor. El lenguaje más usual encontrado en los ambientes procedurales de un sistema de televisión digital es Java. El Ginga-NCL (o Máquina de Presentación) es un subsistema lógico del Sistema Ginga que procesa documentos NCL. Un componente-clave del Ginga-NCL es el mecanismo de decodificación del contenido informativo (NCL *formatter*). Otros módulos importantes son el usuario basado en XHTML, que incluye un lenguaje de estilo (CSS) e intérprete ECMAScript, y el mecanismo LUA, que es responsable por la interpretación de los *scripts* LUA.

El Ginga-J (o Máquina de Ejecución) es un subsistema lógico del Sistema Ginga que procesa aplicaciones procedurales (Xlets Java). Un componente clave del ambiente del aplicativo procedural es el mecanismo de ejecución del contenido procedural, que tiene por base una máquina virtual Java.

Es importante observar que en una implementación únicamente Ginga-NCL o únicamente Ginga-J, ya sea en receptores fijos o móviles, está prohibida la reivindicación de cualquier tipo de conformidad con el SBTVD. Esto garantiza que el Ginga ofrece perfiles siempre compatibles con versiones anteriores.

Decodificadores comunes de contenido deben servir para las necesidades de aplicativos tanto procedurales como informativos de decodificación y presentación de contenidos comunes del tipo PNG, JPEG, MPEG y otros formatos. El Ginga-Core está compuesto por decodificadores y procedimientos comunes de contenido para obtener contenidos transportados en flujos de transporte MPEG-2 (TS) y a través de un canal de retorno. El Ginga-Core también debe soportar el modelo de exhibición conceptual descrito en la ABNT NBR 15606-1.

La arquitectura (ver Figura 1) y las facilidades de la especificación Ginga se deben destinar a la aplicación en sistemas y receptores de transmisión para transmisión terrestre (*over-the-air*). Además, la misma arquitectura, y facilidades, se pueden aplicar a otros sistemas de transporte (como sistemas de televisión vía satélite o cable).

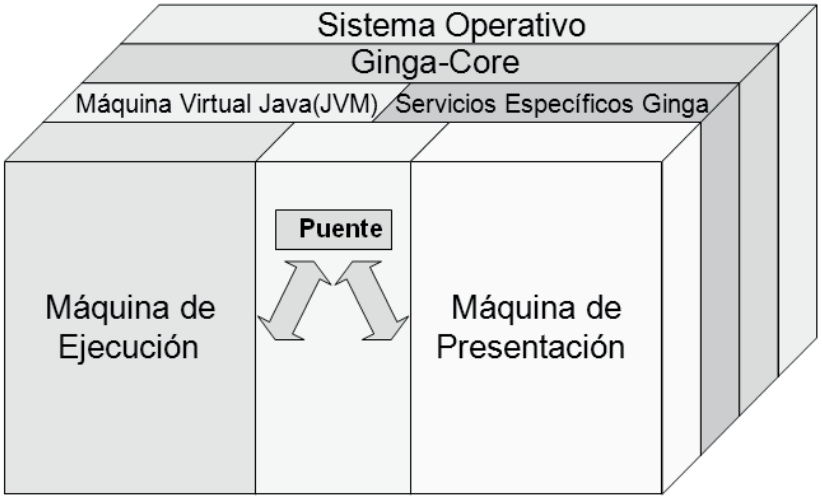

Figura 1 – Arquitectura en alto nivel del *middleware* Ginga

5.2 Arquitectura Ginga-J

5.2.1 Contexto

La Figura 2 presenta el contexto en que la pila del *software* Ginga-J es ejecutada. El Ginga-J es una especificación de *middleware* distribuido, que reside en un dispositivo Ginga (dispositivo que embarque el *middleware* Ginga - un receptor de televisión digital).

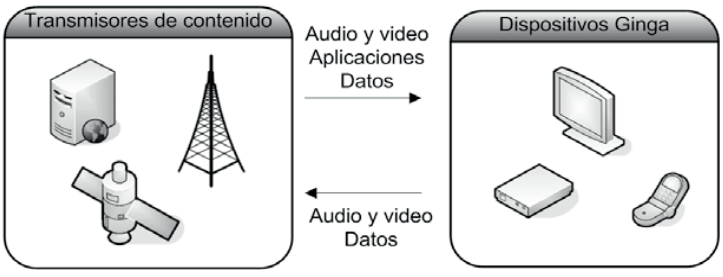

Figura 2 – Contexto del Ginga-J

El dispositivo Ginga debe tener acceso a flujos de vídeo, audio, datos y otros recursos de media, que deben ser transmitidos a través del aire, cable, satélite o a través de redes IP. Las informaciones recibidas deben ser procesadas y presentadas a los televidentes.

5.2.2 Arquitectura

El modelo Ginga-J distingue entre las entidades y recursos de *hardware*, *software* del sistema y aplicativos conforme a lo descrito en la Figura 3.

Las aplicaciones residentes pueden ser implementadas usando funciones no estandarizadas, provistas por el Sistema Operativo del dispositivo de Ginga, o por una implementación particular del Ginga. Los activos residentes también pueden incorporar funcionalidades provistas por las API estandarizadas Ginga-J. Los aplicativos transmitidos (Xlets) siempre deben utilizar API estandarizadas provistas por el Ginga-J.

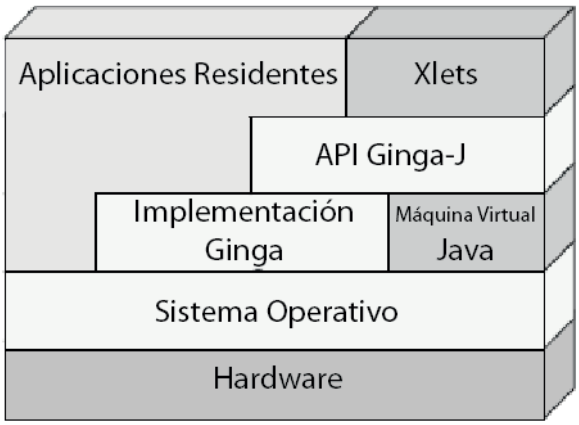

Figura 3 – Arquitectura Ginga-J y ambiente de ejecución

En general, el Ginga es ajeno a cualesquier aplicativos residentes. Estas aplicaciones residentes incluyen, pero no se limitan a: *closed caption*, mensajes del sistema de acceso condicional (*Conditional* Access – CA), menús del receptor y guías de programación electrónica (*Eletronic Program Guide* – EPG) residente.

Los aplicativos residentes pueden tener prioridad sobre los aplicativos Ginga. Como ejemplo, el *closed caption* y mensaje de emergencia deben tener prioridad en el Sistema Ginga.

6 Formato del contenido

El formato del contenido para el sistema brasileño de televisión digital terrestre debe estar de acuerdo con la ABNT NBR 15606-1.

7 Modelo de aplicación Ginga-J

7.1 Modelo de aplicación

7.1.1 Ciclo de vida

El modelo de aplicación Ginga-J debe estar de acuerdo con el modelo de aplicación definido en JAVADTV 1.3:2009. De esta forma, aplicaciones Ginga-J deben contener una clase implementando la interfaz *javax.microedition.xlet.Xlet* (ver PBP 1.1:2008), que debe ser referenciada de acuerdo con las definiciones de señalización de aplicación (ver ABNT NBR 15606-3:2007, 12.16). De lo contrario, la clase (y la instancia de la aplicación) puede ser ignorada.

Aplicaciones Ginga-J deben ser ejecutadas en un ambiente orientado a servicios y mantenidas por un gestor de aplicaciones global del sistema. Todo servicio se presenta en un contexto de servicio, que podría ser definido como su ambiente de ejecución. Para aplicaciones Ginga-J, el contexto de servicio es representado por una instancia de la clase *javax.microedition.xlet.ServiceContext.*

Además, aplicaciones Ginga-J se pueden controlar tanto por un *zapper*, por la emisora (ver ABNT NBR 15606-3:2007, 12.16), por otra aplicación Ginga-J usando la API "*Application Management and LifeCycle Control*" (ver JAVADTV 1.3:2009) o incluso por documentos Ginga-NCL. Considerando que la aplicación providencia una clase que implementa la interfaz *javax.microedition. xlet.Xlet*, esta clase contiene al menos cuatro métodos que son llamados por la plataforma para informar a la aplicación de cambios de ciclo de vida inminentes. Un diagrama exhibiendo el ciclo de vida de aplicaciones Ginga-J se muestra en la Figura 4.

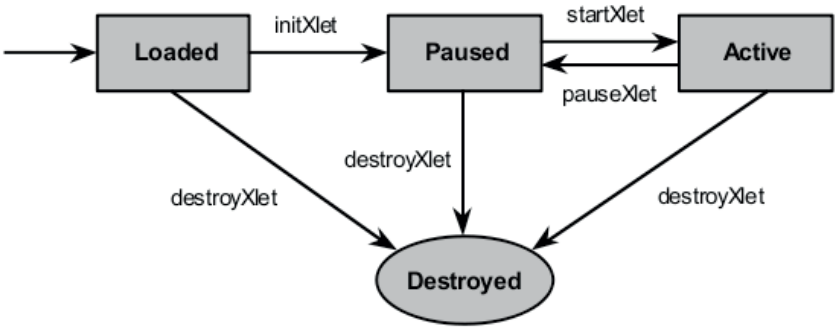

Figura 4 – Diagrama con estados del ciclo de vida de un Xlet

Inicialmente, después que se obtengan los datos de la aplicación, es creado el objeto que implementa *javax.microedition.xlet.Xlet* usando su constructor. Si el constructor estándar vuelve sin disparar una excepción, la instancia de la aplicación será considerada como en el estado "Loaded", sino la instancia de la aplicación será considerada como en el estado "Destroyed" y descartada.

NOTA El inicio de los recursos utilizados por la aplicación se hace en el método *initXlet*() y no en el constructor de la clase. La llamada al constructor es dependiente de implementación.

Para iniciar la aplicación del método *initXlet* se llama a un objeto instancia de *javax.microedition. xlet.XletContext*, que posee información del contexto de ejecución para la aplicación, incluyendo propiedades y mecanismos para notificación de cambios de estados iniciados por la aplicación. Así que la instancia de la aplicación se haya cargado e instado con éxito, el gestor de aplicaciones puede cambiar el estado de la instancia de aplicación para "Paused".

NOTA Es posible que el método *initXlet* sea llamado de forma asíncrona.

El método *startXlet* puede entonces ser llamado para informar a la aplicación que será convertida para el estado "Active", iniciando su ejecución.

El método *pauseXlet* puede ser llamado para informar a la aplicación que debe moverse para el estado "Paused" y que debe minimizar su consumo de recursos. La aplicación puede moverse de vuelta para el estado "Active" tras una nueva llamada al método *startXlet*. Una instancia de aplicación en el estado "Paused" debe reducir su consumo de recursos si tiene la intención de maximizar su probabilidad de supervivencia. Esto no implica que no pueda mantener cualesquier recursos, pero en el caso de que los mantenga, debe tener una prioridad menor al acceso de recursos como si estuviese en el estado "Active".

El método *destroyXlet* puede ser llamado en cualquier estado y se usa para avisar a la aplicación que está lista para terminar su ejecución. La aplicación debe salvar la información de su estado (si fuera posible y necesario) y liberar recursos previamente utilizados lo más rápido posible. Este método posee un parámetro booleano que indica si es incondicional el que esta aplicación deba ser parada. En el supuesto de que la aplicación esté siendo parada debido a una operación de selección de servicio, la finalización de la aplicación será incondicional (ver sección 7.1.3).

NOTA Una instancia de aplicación puede entrar en este estado apenas una vez.

Si un método en la interfaz Xlet dispara *javax.microedition.xlet.XletStateChangeException* (ver PBP 1.1:2008), por estándar el Xlet permanece en el estado en que estaba inmediatamente antes de la llamada que disparó la excepción. La única excepción a esta regla es el método *destroyXlet* cuando el parámetro booleano incondicional fuera pasado con valor verdadero. En este escenario, disparar XletStateChangeException no puede producir ningún efecto y el Xlet debe ser destruido.

El método *initXlet* debe ser llamado apenas una vez. Además, el gestor de aplicaciones puede elegir mover el Xlet al estado “Destroyed” (sin llamar destroyXlet) algún tiempo específico después de la implementación, en caso que sea disparada una XletStateChangeException.

Las aplicaciones en el estado “Destroyed” no pueden ser iniciadas usando la API “*Application Management and LifeCycle Control* “ (ver JAVADTV 1.3:2009) o cualquier otro mecanismo disponible en la plataforma.

7.1.2 Inicio de aplicaciones

Cuando un nuevo servicio es seleccionado para exhibición el gestor de aplicaciones global del sistema debe verificar las aplicaciones disponibles de acuerdo con la ABNT NBR 15606-3:2007, 12.16. En particular, este gestor también identifica aplicaciones que deben ser iniciadas inmediatamente tras la selección del servicio en cuestión.

Si ya existe una aplicación en ejecución en el momento en que acontece la selección de servicio que la contiene, ésta puede continuar en ejecución caso sea señalizada como una aplicación válida para el nuevo servicio seleccionado.

Tras la selección de un servicio es posible que aplicaciones alternativas señalizadas sean iniciadas por el usuario, por el *zapper* o por cualesquier otras aplicaciones usando la API *“Application Management and LifeCycle Control”* (ver JAVADTV 1.3:2009). Es decir, el usuario puede iniciar una aplicación tras recibir una oferta de aplicaciones a través de alguna interfaz de usuario. Ya que tal interfaz es dependiente de implementación, servicios señalizados deben indicar explícitamente caso necesiten que la aplicación sea iniciada automáticamente (ver ABNT NBR 15606-3:2007, 12.16).

7.1.3 Finalización de aplica

Aplicaciones Ginga-J pueden voluntariamente terminar su ejecución usando la API Xlet (ver PBP 1.1:2008) o pueden ser finalizadas por el gestor de aplicaciones global del sistema.

Además, una aplicación debe tener su ejecución interrumpida incondicionalmente siempre que acontezca una de las siguientes condiciones:

— la tabla AIT (ver ABNT NBR 15603:2007) fue actualizada o fue cambiada y en esta nueva versión no consta referencia para la aplicación en cuestión;

— la tabla AIT (ver ABNT NBR 15603:2007) ya no es más referenciada en la tabla PMT (ver ABNT NBR 15603:2007) del servicio que está siendo exhibido.

Cuando una instancia de aplicación se elige para que tenga su ejecución terminada, el gestor de aplicaciones debe llamar su método *destroyXlet*. Conforme lo descrito en 7.1.1, caso esta instancia se esté parando debido a una operación de selección de servicio, su finalización debe ser incondicional.

7.1.4 Soporte a múltiples aplicaciones

Las aplicaciones Ginga-J deben ejecutarse en un ambiente multitarea orientado a eventos de media señalizados por difusión y entradas de eventos del usuario. El modelo de aplicación ha sido proyectado para ser extensible. Es posible soportar múltiples aplicaciones competidoras que están cooperando (proyectadas para comunicarse entre sí y compartir recursos) o no (independientes y compitiendo por recursos).

El modelo de soporte a múltiples aplicaciones del Ginga-J debe estar de acuerdo con el modelo de aplicación definido en JAVADTV 1.3:2009. De tal manera, una aplicación no se puede iniciar si una instancia de esta aplicación ya está activa en el servicio seleccionado para exhibición. Para los casos en que más de un Xlet esté en ejecución, no se permiten cualesquier acciones que puedan afectar el estado global de la plataforma (ver 7.1.1).

Se considera cada instancia de aplicación como si estuviese ejecutando en su propia instancia de máquina virtual. No obstante, es de responsabilidad de la emisora garantizar que las aplicaciones ejecutadas simultáneamente en un eventual servicio sean comprensibles al usuario y no causen problemas perceptibles por interferencia mutua.

7.1.5 Compartimiento de recursos entre aplicaciones

Permitir la ejecución simultánea de múltiples aplicaciones implica que deben definirse algunas reglas para que esas aplicaciones compartan recursos disponibles en el sistema. En particular, las aplicaciones en ejecución deben compartir el "Input Focus" y el "Output Focus".

Una aplicación posee el "Input Focus" si, y solamente si, el *java.awt.Component o el com.sun.dtv.lwuit.Component* que posee “Input Focus”, pertenece al árbol de componentes de aquella aplicación. “Input Focus” puede ser requerido por aplicaciones llamando al método *requestFocus* en una de las clases mencionadas anteriormente, dependiendo del tipo de componente gráfico utilizado.

La aplicación que posee el “Input Focus” es, en principio, capaz de recibir eventos de entrada de usuario. Otras aplicaciones que no poseen el “Input Focus” pueden requerir la recepción de un subconjunto de los eventos de entrada de usuario a través de la API “*TV Specific UI functionality Event*” (ver JAVADTV 1.3:2009).

7.1.6 Controlando aplicaciones

Es posible controlar el ciclo de vida de una aplicación a través de la API "*Application Management and LifeCycle Control*” (JAVADTV 1.3:2009), que provee medios para permitir que aplicaciones requieran el *Application Manager* iniciar, parar, pausar y retomar otras aplicaciones.

7.1.7 Comunicación entre aplicaciones

El modelo de comunicación entre aplicaciones del Ginga-J debe estar de acuerdo con el modelo de aplicación definido en JAVADTV 1.3:2009. Las aplicaciones deben usar los mecanismos definidos en la API "Inter-Xlet Communication" (ver PBP 1.1:2008) para ello. La comunicación entre una aplicación y otra es establecida por la conexión entre un objeto a un nombre en el *javax.microedition.xlet.ixc.IxcRegistry* (ver PBP 1.1:2008) y otra aplicación buscando este nombre e invocando los métodos del objeto. Los posibles “namespaces” para registro deben estar de acuerdo con las definiciones establecidas por la interfaz de comunicación entre aplicaciones de JAVADTV 1.3:2009.

7.1.8 Propiedades del ambiente

Además de las propiedades ya definidas por el modelo Ginga-NCL (ver ABNT NBR 15606-2:2007, 7.2.4), el modelo de aplicación Ginga-J debe proveer cada aplicación con un *javax.microedition.xlet.XletContext* (ver PBP 1.1:2008) incluyendo un conjunto de propiedades específicas ya definidas en JAVADTV 1.3:2009, las cuales son mostradas en la Tabla 1.

Tabla 1 – Propiedades del ambiente Ginga-J

Nombre	Tipo de datos	Descripción
com.sun.dtv.persistent.root	Alfanumérico	Directorio base para almacenamiento persistente en la plataforma
com.sun.dtv.orgid	Numérico	Identificador único para la organización responsable por la aplicación. Debe ser el mismo valor transmitido en el campo *organization_id del* descriptor de identificador de aplicaciones (ver ABNT NBR 15606-3:2007, 12.7)
com.sun.dtv.appid	Numérico	Identificador único para la aplicación. Debe ser el mismo valor transmitido en el campo *application_id del* descriptor de identificador de aplicaciones (ver ABNT NBR 15606-3:2007, 12.7)
com.sun.dtv.version	Alfanumérico	El número de versión de la especificación JavaDTV implementada por la plataforma (JAVADTV 1.3:2009)
br.org.ginga.system.version	Alfanumérico	El número de versión de la especificación Ginga-J implementada por la plataforma

7.1.9 Códigos de control de aplicación

El control dinámico del ciclo de vida de aplicaciones es señalizado a través del campo "application_control_code" para la aplicación en la AIT (Ver ABNT NBR 15606-3:2007). Los códigos de control de las aplicaciones Ginga-J se muestran en la Tabla 2.

Tabla 2 – Códigos de control de aplicaciones Ginga-J

Código	Identificador	Descripción
0x00	No definido	Reservado para uso futuro
0x01	AUTOSTART	Aplicativos con el código de control AUTOSTART son iniciados automáticamente, luego enseguida a la selección del servicio que los contiene
0x02	PRESENT	Aplicativos con el código de control PRESENT no pueden ser iniciados automáticamente y deben ser adicionados a la lista de aplicativos disponibles del receptor. Pueden ser inicializados usando la API "*Application Management and LifeCycle Control*" (JAVADTV 1.3:2009)

Tabla 2 (continuación)

Código	Identificador	Descripción
0x03	DESTROY	Aplicativos con el código de control DESTROY deben ser incondicionalmente finalizados por el gestor de aplicaciones. Aplicativos señalizados previamente con código de control STORE deben ser retirados del *cache*
0x04	KILL	Aplicativos con el código de control KILL deben ser finalizados por el gestor de aplicaciones así que sea posible. Caso sea lanzada una excepción del tipo *javax.microedition.xlet.XletStateChangeException* durante un intento de finalización, el aplicativo debe continuar en ejecución Aplicativos señalizados previamente con código de control STORE pueden opcionalmente ser mantenidos en el *cache*
0x05	No definido	Reservado para uso futuro
0x06	REMOTE	Aplicativos con código de control REMOTE no tienen sus archivos transmitidos en el *transport stream* corriente. La fuente de datos para estos aplicativos debe estar de acuerdo con el descriptor de aplicación "*Transport Protocol Descriptor*" (ver ABNT NBR 15606-3:2007) Aplicativos con el código de control REMOTE no pueden ser iniciados automáticamente y deben ser adicionados a la lista de aplicativos disponibles del receptor. Pueden ser inicializados usando la API "*Application Management and LifeCycle Control*" (ver JAVADTV 1.3:2009)
0x07	UNBOUND	Aplicativos con código de control UNBOUND son similares a los aplicativos señalizados con PRESENT, excepto que el usuario decide si la aplicación puede ser almacenada para ejecución posterior. Caso el receptor no posea capacidad de almacenaje disponible para almacenar la aplicación o el usuario elija no instalar la aplicación, ésta debe ser tratada como no disponible (no puede ser listada entre las aplicaciones disponibles).Los receptores sin soporte a almacenaje de aplicaciones deben ignorar las aplicaciones señalizadas con este tipo de control.
0x08	STORE	Aplicativos con código de control STORE no pueden ser iniciados automáticamente, pero indican cuales son las técnicas de *caching* que se pueden utilizar de forma que acelere la carga de sus recursos durante la inicialización (ver 7.2 para otras informaciones) Caso la plataforma no tenga capacidad para realizar técnicas de *caching* proactivo o almacenaje de datos, las aplicaciones señalizadas como STORE deben ser tratadas de manera idéntica a las aplicaciones señalizadas con PRESENT
0x09…0xFF	No definido	Reservado para uso futuro

Caso se reciba un código de control desconocido, la aplicación debe permanecer en el mismo estado en que se encuentra. Cuando un cambio en el código de control provoque un cambio de estado en una aplicación Ginga-J, se debe generar un evento para todas las aplicaciones Ginga-J que estén registradas para recibir notificaciones de cambios en el ciclo de vida de la aplicación en cuestión.

Opcionalmente, la plataforma puede proveer un menú con un listado de aplicaciones señalizadas con STORED, PRESENT y UNBOUND (las que el usuario eligió instalar) para que el usuario decida el momento exacto de iniciarlas.

Salvo aquellas aplicaciones señalizadas con el campo service_bound_flag iniciado con 0, todas las aplicaciones deben ser destruidas durante un cambio de servicio. La ejecución de aplicaciones señalizadas con el campo service_bound_flag dirigido para 0 no se restringe a un servicio específico y no se puede interrumpir durante la selección entre varios servicios (ver 7.3.4 para más detalles). En el cambio de servicio el receptor siempre puede interrumpir la ejecución de estas aplicaciones, caso exista la necesidad de liberación de recursos.

7.2 Almacenamiento y *caching* de aplicaciones

7.2.1 Modelos de almacenamiento

El modelo de aplicación Ginga-J permite que las aplicaciones sean almacenadas e iniciadas a partir de memoria persistente. Las aplicaciones pueden sufrir *caching* proactivamente o en respuesta a una requisición de otra aplicación, sujeto a consentimiento del usuario y límites de recursos de la plataforma. En ambos casos el objetivo es mejorar la velocidad de carga de la aplicación.

Así, puede ser útil para aplicaciones en las que sea deseable tener una exhibición rápida, evitando la latencia al inicio de la misma debido al tiempo de descarga. Estas aplicaciones pueden ser relacionadas a la difusión, caso en que el ciclo de vida de la aplicación permanece controlado por la difusión de la AIT, o puede ser completamente autosuficiente, en cuyo caso la entrada de la AIT para la aplicación es almacenada junto con los datos de la aplicación.

Aplicaciones señalizadas con el código de control STORE o UNBOUND en la AIT deben adicionar información extra en el archivo MANIFEST.MF de manera que indique cuáles archivos deben ser almacenados. Para cada archivo que debe ser almacenado, la entrada en el MANIFEST.MF para este archivo debe contener el valor *true* para el atributo “Persistent-Flag” y un valor para el atributo “File-Version” indicando la versión del archivo. Los receptores que soporten el almacenamiento de aplicaciones deben verificar la versión del archivo almacenado y actualizarlo cuando la aplicación recibida sea mayor. Además, deben adicionarse al MANIFEST.MF entradas de archivo para el *application_id* y el o*rganization_id* ambos señalizados en la AIT para la aplicación en cuestión.

Las aplicaciones almacenadas deben ser visibles por la API “*Application Management and LifeCycle Control*” (JAVADTV 1.3:2009), de la misma manera como cualesquier otras aplicaciones.

7.2.2 Cuestiones de almacenamiento

El espacio disponible para el almacenamiento de las aplicaciones puede no ser suficiente para comportar todas las aplicaciones señalizadas como persistentes (STORE o UNBOUND) siendo así necesario elegir cuáles de ellas deben ser efectivamente persistidas. Eventualmente, también puede ser necesario retirar aplicaciones previamente almacenadas. En estos casos, queda a criterio de la implementación del fabricante del receptor la política de persistencia y remoción de las aplicaciones señalizadas como STORE. Las aplicaciones señalizadas como UNBOUND deben ser instaladas y retiradas con autorización explícita del usuario y deben tener prioridad en relación a las aplicaciones señalizadas como STORE.

Cuando la emisora señalice una nueva versión de una aplicación almacenada, el *middleware* puede sobrescribir la versión antigua de la aplicación con una nueva versión. El momento en que esto acontece no es previsible. Si la versión antigua está en ejecución en el momento de la actualización, la plataforma puede decidir almacenar ambas versiones hasta que la copia actualmente en ejecución termine. En este momento, la copia antigua es retirada del almacenaje. Si la plataforma elige borrar la copia antigua antes que la aplicación termine, el comportamiento de la aplicación en ejecución debe permanecer el mismo (en la práctica esto significa que todas las clases y recursos deben estar cargados en la memoria antes que la versión antigua sea apagada).

7.2.3 *Caching* proactivo

Cuando una aplicación es señalizada con código de control STORE, se permite que la plataforma almacene proactivamente cualesquier archivos que estén indicados en el archivo de descripción de aplicación.

Sin embargo, no es obligatorio cumplir las requisiciones de prioridad caso se utilicen técnicas de almacenamiento proactivo de aplicaciones. En particular, no es obligatorio que el *caching* proactivo almacene todos los archivos con prioridad crítica.

7.3 Transmisión de aplicaciones

7.3.1 Reglas de señalización

Las reglas estándar de señalización de aplicaciones Ginga-J deben obligatoriamente estar de acuerdo con lo descrito en 7.1.9 y con la ABNT NBR 15606-3:2007, 12.16.

7.3.2 Empaquetamiento de aplicaciones

Aplicaciones Ginga-J deben ser empaquetadas, autenticadas y autorizadas de acuerdo con las definiciones especificadas en JAVADTV 1.3:2009. Cada aplicación Ginga-J puede contener uno o más archivos JAR (ver JAR:2009 y JVM:1997). El archivo JAR principal contiene los archivos de clase de la aplicación, archivos de recurso y un manifiesto que describe la aplicación y sus requisitos. El mecanismo de firma de JAR permite que el JAR sea autenticado.

Si la transmisión se hace con uso del carrusel de objetos, no se puede utilizar el empaquetamiento en un archivo JAR, pero obligatoriamente el sistema de archivos transmitido debe estar organizado de la misma manera.

Si la aplicación es transmitida por el canal de interactividad es obligatorio que ésta sea transmitida en archivos JAR.

7.3.3 Autenticación de aplicación

La autenticación de aplicaciones Ginga-J debe estar de acuerdo con la Norma Brasileña vigente en el momento de la transmisión de la aplicación.

NOTA A Norma Brasileña sobre autenticación de aplicación está en elaboración.

7.3.4 Señalizando la misma aplicación en diversos servicios

Para que una aplicación sea considerada como señalizada en diversos servicios, deben cumplirse las siguientes condiciones:

- la señalización debe estar presente en una tabla AIT en todos los servicios;

- el identificador de aplicación debe ser igual en todos los servicios;
- el descriptor de protocolo de transporte debe ser igual en todos los servicios o la aplicación está señalizada con el código de control UNBOUND y ya se encuentra persistida en el receptor.

Si, además de las condiciones descritas anteriormente la aplicación, también está señalizada con el campo *service_bound_flag* con el valor 0, esta aplicación se debe mantener en ejecución entre todos los cambios de servicio, a menos que haya alguna restricción de recursos en el receptor.

7.3.5 *Download* de aplicaciones a través del canal interactivo

Las reglas de transporte de aplicación deben obligatoriamente estar de acuerdo con la ABNT NBR 15606-3.

Las aplicaciones obtenidas a través del canal de interactividad deben estar contenidas en un archivo JAR (ver 7.3.1). El soporte a la compresión en el archivo JAR es opcional.

8 Plataforma Ginga-J

8.1 Plataforma Java

La plataforma Java utilizada para ejecución de aplicaciones Ginga-J es definida de acuerdo con el PBP 1.1:2008.

Los *bytecodes* Java ejecutados por la plataforma deben tener versiones entre 45.3 y 47.

8.2 Consideraciones básicas de la plataforma

8.2.1 Ambiente de ejecución

Cada aplicación Ginga-J debe ser procesada en un ambiente de ejecución aislado, es decir, debe haber una entidad de sistema que represente una JVM en donde cada aplicación debe ser ejecutada sin que haya interferencia en la ejecución de cualquier otra aplicación. Para ello, este ambiente de ejecución debe permitir que cada aplicación sea ejecutada por medio de un cargador (*ClassLoader*) propio o incluso una jerarquía propia de cargadores para accesar clases que no sean parte de la plataforma base del Ginga-J.

Las aplicaciones competidoras no deben compartir directamente instancias de objetos definidas por éstas. Cualquier interacción entre aplicaciones debe ser posible apenas por medio de API específica. Cada ambiente de ejecución debe establecerse al momento en que la aplicación se inicie y descargado luego que la aplicación se destruya. Tras la finalización de cada aplicación, el Gestor de Aplicaciones debe asegurarse de que se ejecuten los finalizadores de clases contenidas en la aplicación.

Las aplicaciones Ginga se pueden ejecutar en modo competencia. Una vez cargada e iniciada una instancia de una aplicación, está prohibida la creación o iniciación de otra instancia de la misma aplicación. Las aplicaciones son diferenciadas por medio de sus respectivos valores de *organization_id* y *application_id* (ver ABNT NBR 15606-3:2007).

Las aplicaciones Ginga-J no pueden sincronizar en clases de sistema o incluso otras instancias estáticas de sistema. Caso contrario, el comportamiento esperado es indefinido.

8.2.2 Jerarquía de paquetes y clases

Apenas métodos y campos (y sus respectivas dependencias) de las clases listadas en esta parte de la ABNT NBR 15606 deben obligatoriamente estar presentes en una implementación Ginga. Aparte de esto, en donde haya dependencia de un paquete específico, su inclusión completa es permitida, pero no obligatoria. El comportamiento para clases y métodos adicionales no se especifica para aplicaciones enviadas por radiodifusión.

Las clases, interfaces y métodos listados o referenciados por este documento y que estén marcados como obsoletos (*deprecated*), deben tener sus marcaciones sobrescritas de manera que esas clases, interfaces o métodos se vuelvan obligatorios en esta parte de la ABNT NBR 15606. Es altamente recomendado que las aplicaciones Ginga-J no utilicen estos elementos depreciados, ya que pueden dejar de ser soportados en versiones futuras de esta parte de la ABNT NBR 15606.

La inclusión de algún paquete por esta parte de la ABNT NBR 15606 no implica directamente la inclusión de sus subpaquetes.

Las aplicaciones Ginga-J no pueden definir clases o interfaces en cualquier paquete (o *namespace*) definido en esta parte de la ABNT NBR 15606.

Las clases de implementación de la plataforma Ginga-J no pueden estar contenidas en el paquete vacío (*default package*).

8.2.3 Notificación de eventos

Para todas las clases listadas en esta parte de la ABNT NBR 15606 donde haya métodos para registro/desregistro de notificaciones, sucesivas llamadas para registro de oyentes (*listeners*) deben tener el mismo efecto que una sola llamada. Luego entonces, cada evento debe ser notificado apenas una vez por oyente. Además, una requisición de cancelación de registro no puede surtir efecto en caso que el oyente en cuestión no esté registrado.

El número de procesos (*threads*) utilizados para notificación de eventos a los oyentes depende de implementación. Las aplicaciones Ginga-J no pueden bloquear el procesamiento en sus oyentes de forma que evite que otros oyentes no sean notificados.

Todas las clases de eventos listados en esta parte de la ABNT NBR 15606 deben extender la clase *java.util.EventObject*.

8.2.4 Codificación de texto

La codificación de texto estándar para la plataforma Ginga-J debe ser UTF-8 conforme el método *java.io.DataOutput.writeUTF* (ver PBP 1.1:2008). El estándar “Latin1” también debe ser soportado de acuerdo con ISO/IEC 8859-1:1998.

8.2.5 Ciclo de vida de las aplicaciones

La máquina de estados definida en 7.1.1 debe funcionar de manera que el comportamiento de las aplicaciones respete las siguientes restricciones:

— la latencia percibida durante la iniciación de la aplicación debe ser la mínima posible;

— una aplicación puede ser destruida en cualquier momento.

El Gestor de Aplicaciones debe usar la Xlet API (ver PBP 1.1:2008) para ordenar cambios en el ciclo de vida de las aplicaciones. Luego, varios factores pueden estimular al Gestor de Aplicaciones, como por ejemplo:

— señalizaciones originadas en las emisoras (ver 7.1.9);

— selección a partir de menú propietario con lista de aplicaciones;

— orden originada en otra aplicación Ginga-J por medio de la API "Application Management and LifeCycle Control" (JAVADTV 1.3:2009) (ver 7.1.6);

— orden originado en documento NCL embarcando uno o más Xlets.

La propia aplicación puede decidir cambiar su estado. Para ello, debe usar su instancia de *javax.microedition.xlet.XletContext* para requerir ese cambio al Gestor de Aplicaciones.

8.3 Infraestructura común

Las API definidas en (ver CDC 1.1:2008; FP 1.1:2008 y PBP 1.1:2008) son incluidas por esta especificación como base para el funcionamiento de la plataforma Ginga-J.

Los paquetes específicos para televisión digital definidos en JAVATV 1.1:2008 también deben ser empleados en esta parte de la ABNT NBR 15606. Complementariamente, deben respetarse las siguientes restricciones:

— el grupo de procesos (threads) de una aplicación Ginga-J no puede contener procesos con prioridad mayor que java.lang.Thread.NORM_PRIORITY;

— el retorno del método System.currentTimeMillis debe ser sincronizado con la fecha y hora transmitida en la TOT;

— el retorno del método System.currentTimeMillis debe tener granularidad menor o igual a 10 ms;

— el TimeZone utilizado por la JVM debe estar de acuerdo con las configuraciones del receptor.

— el java.util.Calendar debe ser sincronizado de acuerdo con la fecha y hora transmitida en la TOT;

— el java.util.Locale estándar de la JVM debe ser definido como "pt_BR";

— las aplicaciones Ginga-J no pueden utilizar el método java.util.TimeZone.setDefault. El comportamiento de este método depende de implementación;

— los flujos de salida System.out y System.err deben estar disponibles para aplicaciones Ginga-J. Sin embargo, el flujo de entrada System.in no puede estar disponible;

— los métodos Runtime.traceInstructions y Runtime.traceMethodCalls deben estar disponibles sin que esto tenga impacto negativo en la ejecución de las aplicaciones y sin interferencia en el funcionamiento de otras API;

— el método System.gc depende de implementación y no tiene comportamiento definido;

— el método Runtime.gc depende de implementación y no tiene comportamiento definido;

— la plataforma debe utilizar mecanismos previstos en java.lang.SecurityManager. checkPackageDefinition y java.lang.SecurityManager.checkPackageAccess para evitar el uso indebido de clases de sistema por parte de las aplicaciones;

— el paquete javax.tv.xlet es considerado obsoleto (deprecated) por esta especificación. Sin embargo, de manera que permita una mayor integración con aplicaciones legadas, instancias de javax.microedition.xlet.XletContext deben implementar también javax.tv.xlet.XletContext. Ambas interfaces contienen métodos y comportamientos semejantes. Se debe evitar el uso de la interfaz javax.tv.xlet.XletContext por las aplicaciones Ginga-J;

— el paquete javax.tv.graphics es considerado obsoleto (deprecated) por esta especificación. Las clases e interfaces definidas en él no pueden ser utilizadas por las aplicaciones Ginga-J;

— la granularidad mínima para la clase javax.tv.util.TVTimer debe ser menor o igual a 10 ms;

— el menor intervalo de repetición para la clase javax.tv.util.TVTimer debe ser 40 ms;

— llamadas al método javax.tv.util.TVTimer.scheduleTimerSpec deben provocar excepciones del tipo TVTimerScheduleFailedException caso no haya temporizadores (timers) disponibles en el sistema.

8.4 Presentación gráfica y tratamiento de eventos

8.4.1 LWUIT, *LightWeight user interface toolkit*

Esta parte de la ABNT NBR 15606 utiliza JAVADTV 1.3:2009 para definir los componentes gráficos y el mecanismo de tratamiento de eventos de usuario. Las aplicaciones disponen de funcionalidades gráficas como:

— componentes gráficos de alto nivel;

— aplicación de temas personalizables a los componentes gráficos;

— tratamiento de jerárquico a través de contenedores y componentes;

— abstracción de componentes nativos al sistema.

8.4.2 Interfaz gráfica de usuario

La interfaz gráfica del usuario se hará utilizando los componentes específicos para aplicaciones de televisión y de los componentes gráficos provistos por la LWUIT 1.1:2008 incorporada al JAVADTV 1.3:2009. Los paquetes que definen estos componentes gráficos son:

- com.sun.dtv.ui – define los componentes gráficos específicamente vinculados a televisión;
- com.sun.dtv.lwuit – contiene los componentes gráficos que dan soporte a la creación de interfaces gráficas de usuario;
- com.sun.dtv.lwuit.animations – habilita tanto componentes gráficos como transiciones animadas entre contenedores;
- com.sun.dtv.lwuit.geom – define los elementos geométricos básicos para dibujo;
- com.sun.dtv.lwuit.layouts – define tipos útiles de layouts gráficos;

- com.sun.dtv.lwuit.list – define estructuras de lista personalizables utilizadas en componentes de otros paquetes como el com.sun.dtv.lwuit;
- com.sun.dtv.lwuit.painter – permite dibujar arbitrariamente elementos gráficos a partir de imágenes planas, escaladas y/o tiled;
- com.sun.dtv.lwuit.plaf – permite personalizar la apariencia de los componentes gráficos;
- com.sun.dtv.lwuit.util – paquete de utilidades.

La interfaz com.sun.dtv.ui.MatteEnabled y las clases com.sun.dtv.ui.AnimatedMatte y com.sun.dtv.ui.StaticMatte deben estar presentes y ser implementadas de manera que mantengan el contrato con las aplicaciones. Sin embargo, la funcionalidad de composición gráfica utilizando las informaciones de transparencia provistas por las instancias de estos objetos es opcional para las implementaciones de la plataforma Ginga-J.

8.4.3 Tratamiento de los planos de la plataforma

8.4.3.1 Consideraciones generales

El paquete com.sun.dtv.ui permite el acceso de manera genérica a los planos ofrecidos por la plataforma, para exhibición de contenido organizado en capas en la pantalla del dispositivo. Conforme la ABNT NBR 15606-1, la organización de los planos, o capas, se presenta de acuerdo con la Figura 5.

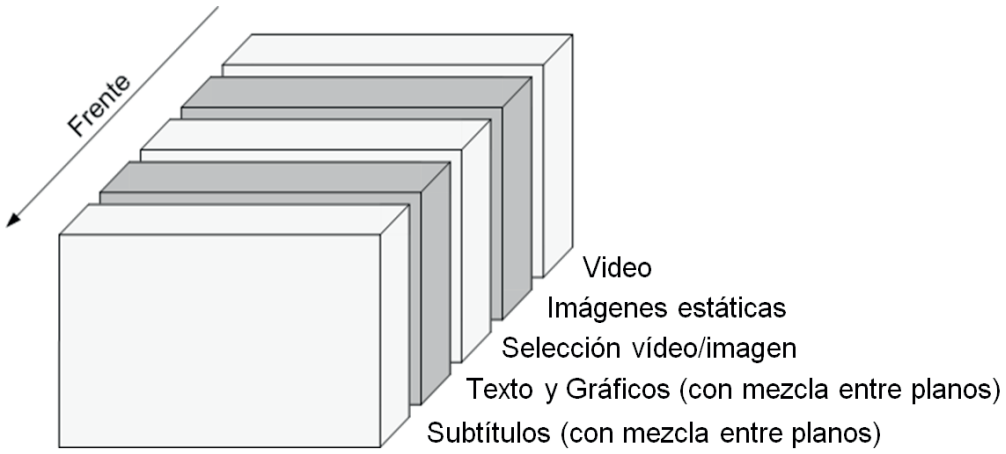

Figura 5 – Estructura de capas para la presentación de servicios

El plano de subtítulos no es accesible por aplicaciones Ginga, siendo una característica nativa del receptor. Restan, por lo tanto, cuatro planos sobre los cuales una aplicación Java puede operar. Siguiendo esta convención, estos planos son retornados siempre de esta manera, en el método *getAllPlanes*() de la clase *com.sun.dtv.ui.Screen* (que a su vez se obtiene a través de la clase *com.sun.dtv.ui.Device*):

— Plane[0]: Plano de texto y gráficos;

— Plane[1]: Plano de selección video/imagen;

— Plane[2]: Plano de imágenes estáticas;

— Plane[3]: Plano de video.

Para cada uno de estos planos, es posible obtener sus características y efectuar operaciones gráficas sobre ellos. Estas características, así como las operaciones soportadas por cada plano, mantienen la adherencia con las definiciones de ABNT NBR 15606-1. Este detalle se hace en las subsecciones abajo, en donde se definen los valores de retorno para cada uno de los métodos, dentro del objeto com.sun.dtv.ui.Capabilities correspondiente al respectivo plano.

8.4.3.2 Plano de texto y gráficos

El plano de texto y gráficos es el plano sobre el cual la aplicación puede dibujar elementos gráficos (primitivas geométricas e imágenes) con alta definición de colores y canal de transparencia, sobre el video.

El retorno del método getID(): “GraphicPlane” está detallado en la Tabla 3.

Tabla 3 – Detalle de funcionalidades para el plano de texto y gráficos

Función	Descripción
getBitsPerPixel()	32
getColorCodingModel()	ColorCoding.ARGB8888
getSupportedPixelAspectRatios()	Un único objeto *Dimension* (1,1) es el valor recomendado (indica píxeles con proporción 1:1). Cualquier valor diferente de éste es considerado opcional
getSupportedPlaneAspectRatios()	Un único objeto *Dimension* con valor de construcción (16,9) o (4,3), dependiendo de la configuración de la salida de video del receptor
getSupportedScreenResolutions()	Un único objeto *Dimension* con las dimensiones del plano de gráficos y textos en el momento
isAlphaBlendingSupported()	*True*
isGIFRenderingSupported()	*True* sólo para plataformas que permiten exhibición de imágenes GIF en el plano gráfico
isGraphicsRenderingSupported()	*True*
isImageRenderingSupported()	*True*
isJPEGRenderingSuppored()	*True*
isPNGRenderingSuppored()	*True*
isRealAlphaBlendingSupported()	*True*
isVideoRenderingSupported()	*True* sólo para plataformas que permiten exhibición de monomedias de video (o incluso un flujo elemental de video) en el plano gráfico
isWidgetRenderingSupported()	*True*

El com.sun.dtv.ui.DTVContainer asociado a este com.sun.dtv.ui.Plane debe soportar todos los tipos de com.sun.dtv.lwuit.component y operaciones gráficas definidas en la API del LWUIT, excepto por la exhibición de tipos de media no permitidos en el plano de texto y gráficos (ver ABNT NBR 15606-1).

8.4.3.3 Plano de selección video/imagen

El plano de selección video/imagen permite definir áreas de precedencia entre el plano de imágenes estáticas y el plano de video. Es decir, en qué áreas rectangulares de la pantalla el plano de imágenes estáticas se exhibirá sobre el plano de video, o viceversa.

El retorno del método getID: “SwitchingPlane” está detallado en la Tabla 4.

Tabla 4 – Detalle de funcionalidades para el plano de video e imágenes

Función	Descripción
getBitsPerPixel()	1
getColorCodingModel()	ColorCoding.ONE_BPP
getSupportedPixelAspectRatios()	Ídem a GraphicPlane
getSupportedPlaneAspectRatios()	Ídem a GraphicPlane
getSupportedScreenResolutions()	Un único objeto *Dimension* con las dimensiones del plano de video en el momento
isAlphaBlendingSupported()	*True* (El plano de selección video/ imagen es el modo por el cual el plano de imágenes estáticas implementa alpha blending)
isGIFRenderingSupported()	*False*
isGraphicsRenderingSupported()	*False*
isImageRenderingSupported()	*False*
isJPEGRenderingSuppored()	*False*
isPNGRenderingSuppored()	*False*
isRealAlphaBlendingSupported()	*False*
isVideoRenderingSupported()	*False*
isWidgetRenderingSupported()	*True*

El com.sun.dtv.ui.DTVContainer asociado a este com.sun.dtv.ui.Plane soporta, además de la definición de un layout manager (método setLayout()), sólo las llamadas addComponent() y removeComponent(), y solamente cuando el com.sun.dtv.lwuit.Component pasado como parámetro sea del tipo br.org.sbtvd.ui.SwitchArea (ver Anexo E). Los demás métodos, aunque no generen error en la aplicación, no tienen efecto, pues son referentes a operaciones no soportadas por el plano de selección video/imagen.

Por medio del *com.sun.dtv.lwuit.plaf.Style* asociado al *com.sun.dtv.ui.DTVContainer* de este plano, se puede definir el contenido de video que se exhibirá por encima del Still Picture Plane o viceversa. Las instancias de com.sun.dtv.lwuit.plaf.Style asociadas al *com.sun.dtv.ui.DTVContainer* del plano de selección video/imagen solamente pueden contener colores sólidos (java.awt.Color). El color negro, java.awt.Color.BLACK, representa que el vídeo se debe exhibir por encima del Still Picture Plane. Con el uso de cualquier otro color, el contenido del Still Picture Plane será exhibido delante del video. De esta forma, las aplicaciones usan el modelo de colores del Java (RGB888 ó ARGB8888) para controlar el plano de selección video/imagen. La implementación de esta API efectuará la conversión del modelo de colores del Java para el modelo de colores del plano de selección video/imagen de acuerdo con la siguiente función:

$$f(x)= \begin{cases} x = \ iava.awt.Color.BLACK \Rightarrow video\ seleccionado \\ x <> java.awt.,Color.BLACK \Rightarrow imagen\ estática\ seleccionada \end{cases}$$

8.4.3.4 Plano de imágenes estáticas

Este es el plano sobre el cual la aplicación puede exhibir imágenes de alta resolución y profundidad de colores en el formato JPEG. Típicamente, es usado para definir un plano de fondo para la aplicación en pantallas en donde el vídeo es redimensionado. Sin embargo, a través del uso conjunto con el plano de selección video/imagen, puede usarse para exhibir imágenes JPEG en alta resolución sobre el video.

El retorno del método getID: “StillPlane” está detallado en la Tabla 5.

Tabla 5 – Detalle de funcionalidades para el plano de imágenes estáticas

Función	Descripción
getBitsPerPixel()	16
getColorCodingModel()	ColorCoding.YUV442
getSupportedPixelAspectRatios()	Ídem a GraphicPlane
getSupportedPlaneAspectRatios()	Ídem a GraphicPlane
getSupportedScreenResolutions()	Un único objeto Dimension con las dimensiones del plano de Imágenes Estáticas en el momento
isAlphaBlendingSupported()	True (implementado a través del plano de selección video/imagen)
isGIFRenderingSupported()	False
isGraphicsRenderingSupported()	False
isImageRenderingSupported()	True
isJPEGRenderingSuppored()	True
isPNGRenderingSuppored()	False
isRealAlphaBlendingSupported()	False
isVideoRenderingSupported()	*False*

Tabla 5 (continuación)

Función	Descripción
isWidgetRenderingSupported()	*True*

El com.sun.dtv.ui.DTVContainer asociado a este com.sun.dtv.ui.Plane soporta, además de la definición de un layout manager (método setLayout()), sólo las llamadas addComponent() y removeComponent(), y solamente cuando el Component pasado como parámetro sea del tipo *br.org.sbtvd.ui.StillPicture* (ver Anexo E). Los demás métodos, aunque no generen error en la aplicación, no tienen efecto, ya que se refieren a operaciones no soportadas por el plano de imágenes estáticas.

Por medio del *com.sun.dtv.lwuit.plaf.Style* asociado a la instancia del objeto *com.sun.dtv.ui.DTVContainer* de este plano, puede definirse un color de fondo sólido (sin transparencia). Este color se debe exhibir como contenido del plano de imágenes estáticas en todas las regiones de este plano que no son rellenadas por objetos del tipo *br.org.sbtvd.ui.StillPicture*. Este color debe definirse en el modelo de colores RGB888, propio del ambiente Java, incumbiendo a la plataforma la conversión para el modelo de colores YUV442.

8.4.3.5 Plano de video

El plano de video exhibe el flujo elemental de video del servicio, u opcionalmente, monomedias de video. El contenido exhibido en este plano puede ser manipulado a través de controles JMF.

El retorno del método getID: “VideoPlane” está detallado en la Tabla 6.

Tabla 6 – Detalle de funcionalidades para el plano de video

Función	Descripción
getBitsPerPixel()	Lanza una SetupException
getColorCodingModel()	Lanza una SetupException
getSupportedPixelAspectRatios()	Ídem a GraphicPlane
getSupportedPlaneAspectRatios()	Ídem a GraphicPlane
getSupportedScreenResolutions()	Un único objeto Dimension con las dimensiones del plano de video en el momento
isAlphaBlendingSupported()	False (Es el plano más al fondo de la cadena)
isGIFRenderingSupported()	False
isGraphicsRenderingSupported()	False
isImageRenderingSupported()	False
isJPEGRenderingSuppored()	False
isPNGRenderingSuppored()	False
isRealAlphaBlendingSupported()	*False*

Tabla 6 (continuación)

Función	Descripción
isVideoRenderingSupported()	*True*
isWidgetRenderingSupported()	*False*

No hay objeto *com.sun.dtv.ui.DTVContainer* asociado a este com.sun.dtv.ui.Plane. El efecto de la invocación de cualquier otro método diferente de *getCapabilities*() en esta instancia en particular, es dependiente de la implementación.

8.4.3.6 Composición de los planos

La Figura 6 presenta un ejemplo de composición entre los diferentes planos. En la composición final se puede observar cómo es exhibido el contenido de video en el área indicada por la región negra en el plano de selección video/imagen. De la misma forma, los contenidos del plano de imágenes estáticas se exhiben delante del video en la región “blanca” del plano de selección video/imagen.

El contenido del plano de gráficos y textos siempre se exhibe por delante de los demás planos.

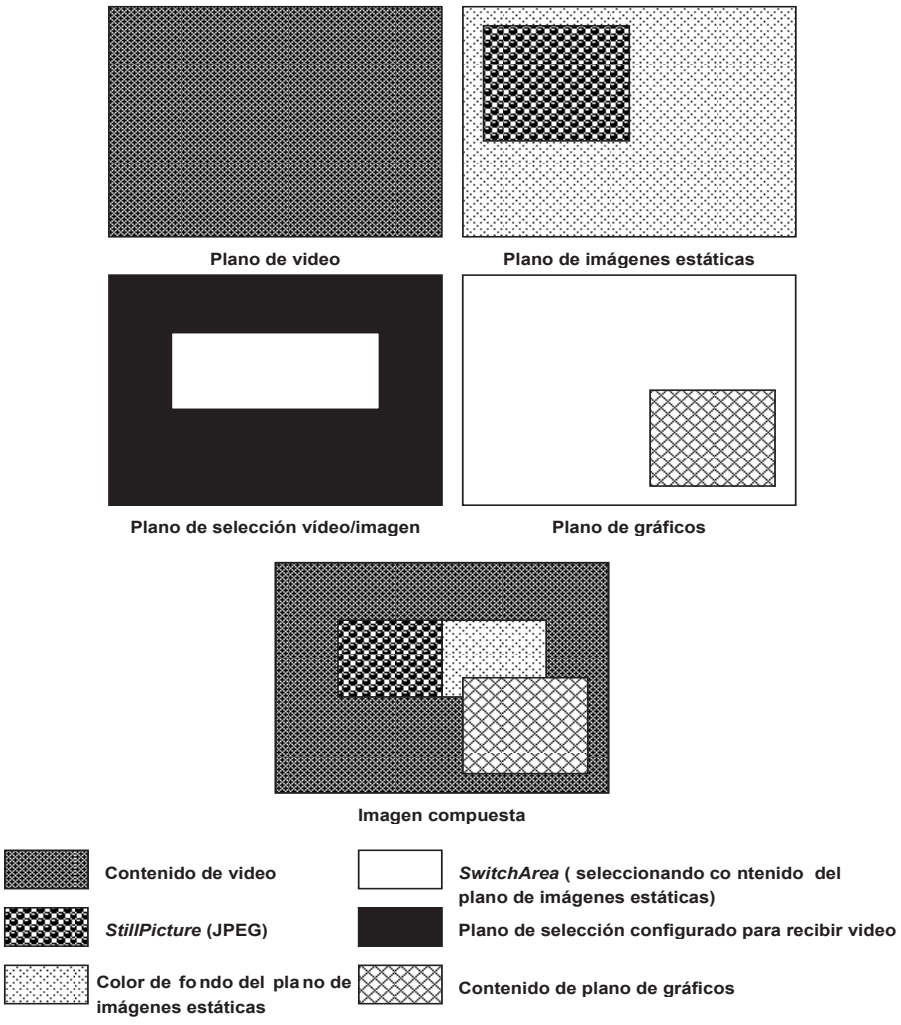

Figura 6 – Ejemplo de composición de los plano de exhibición de contenido

8.4.4 Tratamiento de eventos del usuario

El mecanismo de tratamiento de eventos del usuario es proveído por componentes especializados en televisión y de los componentes proveídos por la LWUIT 1.1:2008 incorporada al JAVADTV 1.3:2009. Los paquetes que definen estos componentes son:

- com.sun.dtv.ui.event – mecanismo para tratamiento de eventos específicos de televisión;
- com.sun.dtv.lwuit.events – mecanismo de tratamiento de eventos relacionados a los componentes gráficos definidos en la LWUIT 1.1:2008 incorporada al JAVADTV 1.3:2009.

Todo el mecanismo de tratamiento de eventos del usuario se basa en el modelo EDT (*Event Dipatch Thread*), dando a la plataforma toda la ciencia de los eventos y repasando a las aplicaciones su tratamiento especializado.

La ABNT NBR 15604:2007, 7.2.28.3, define las funciones mínimas del mando a distancia para receptores que dispongan de mecanismo de interactividad. Estas funciones deben mapearse para los tipos de eventos definidos en la clase com.sun.dtv.lwuit.event.RemoteControlEvent de la JAVADTV 1.3:2009. La Tabla 7 presenta este mapeado.

Tabla 7 – Mapeado de las funciones de la ABNT NBR 15604:2007 en los eventos definidos por la JAVADTV 1.3:2009

Funciones definidas en la ABNT NBR 15604:2007		Clase com.sun.dtv.lwuit.event. RemoteControlEvent
Confirma		VK_CONFIRM
Salir		VK_ESCAPE
Volver		VK_BACK
Direccional	Arriba	VK_UP y VK_KP_UP
	Abajo	VK_DOWN y VK_KP_DOWN
	Izquierda	VK_LEFT y VK_KP_LEFT
	Derecha	VK_RIGHT y VK_KP_RIGHT
Coloridas	Roja	VK_COLORED_KEY_0
	Verde	VK_COLORED_KEY_1
	Amarilla	VK_COLORED_KEY_2
	Azul	VK_COLORED_KEY_3

8.5 Información y selección de servicios

8.5.1 Consideraciones generales

La API de informaciones de servicio de esta especificación es responsable por dar a los aplicativos el acceso a las informaciones presentes en las tablas de informaciones de servicio MPEG. Estas informaciones deben incluir los flujos de audio y video presentes en cada servicio multiplexado, incluyendo también descripción textual de los servicios y eventos que lo componen, entre otros.

Esta API tiene como base la definición de las API de la ARIB STD-B.23:2006, Anexo M, que se

basa en el GEM y provee funcionalidades de acceso a la información de servicios de acuerdo con el modelo de referencia del estándar de televisión digital japonés ARIB STD-B10:2008, lo mismo que el SBTVD extiende. Sin embargo, una vez que esta parte de la ABNT NBR 15606 tiene como núcleo la especificación JAVADTV 1.3:2009, fue necesario realizar alteraciones en los elementos de los paquetes *jp.or.arib.tv.si y jp.or.arib.tv.net* con el objetivo de integrar las funcionalidades de acceso a informaciones de servicio dependientes de protocolo. El Anexo B detalla estas funcionalidades.

8.5.2 Integración con API independiente de protocolo

La API de informaciones de servicio independiente de protocolo debe estar de acuerdo con JAVADTV 1.3:2009, 6.4.

8.6 Presentación y ejecución de medias

La API de flujos de media adopta la Java Media Framework (JMF) 1.0, y opcionalmente puede incorporar API adicionales definidas en la Java Media Framework (JMF) 2.1. La JMF 2.1 es retrocompatible con la JMF 1.0.

8.7 Acceso a datos

8.7.1 Consideraciones generales

Se debe utilizar la API java.io (ver PBP 1.1:2008) para acceso a objetos de datos de modo genérico. Clases e interfaces contenidas en este paquete relacionadas con archivos y sistemas de archivos deben obedecer a la siguiente restricción:

— el método java.io.ObjectInputStream.readLine es marcado como obsoleto (deprecated) por esta parte de la ABNT NBR 15606. Luego no se puede usar por aplicaciones enviadas por difusión.

8.7.2 Acceso a archivos

Para una aplicación enviada por radiodifusión y señalizada en un servicio específico, deben buscarse los objetos de datos accedidos por medio de caminos relativos a partir del camino indicado en el descriptor de localización de la aplicación "*ginga_j_application_location_descriptor*" (ver ABNT NBR 15606-3:2007, 12.18.2), en el campo *base_directory*. El camino definido en el descriptor en cuestión se debe considerar como directorio base de la aplicación.

8.7.3 Protocolo de transporte por radiodifusión

Cuando la aplicación es transmitida vía protocolo de radiodifusión, utilizando los protocolos de Carrusel de Objetos DSMCC o Carrusel de Datos DSMCC, deben ser soportadas las extensiones proveídas por la API "Broadcast file and streams handling" (ver JAVADTV 1.3:2009). Esta API da acceso a archivos y datos internos de *streams* como extensión al conjunto de funcionalidades ya disponibles en el paquete *java.io* (ver PBP 1.1:2008).

Aclaraciones sobre operaciones de entrada/salida para objetos enviados por difusión se pueden encontrar en la documentación de la clase *com.sun.broadcast.BroadcastFile* (ver JAVADTV 1.3:2009), así como en JAVADTV 1.3:2009, 8.2.7.

Las aclaraciones antedichas se deben aplicar también a operaciones con objetos de datos del tipo *java.io.File* (ver PBP 1.1:2008).

8.7.4 Almacenamiento persistente

Para fines de almacenamiento persistente se debe disponibilizar el paquete *java.io* (ver PBP 1.1:2008), así como la API “Persistent storage access rights and properties” (ver JAVADTV 1.3:2009).

La propiedad “*com.sun.dtv.persistent.root*” debe ser accesible por medio del método *java.lang.System.getProperty* (ver PBP 1.1:2008) y debe identificar el directorio raíz para almacenamiento persistente. Caminos relativos no se pueden utilizar para accesar objetos en almacenamiento persistente, bajo pena de funcionamiento indefinido en diversas plataformas. La estructura de directorios prevista para almacenamiento persistente se describe en detalles en JAVADTV 1.3:2009, 6.5.

El acceso a archivos o directorios por encima del directorio base de la aplicación debe siempre resultar en una excepción del tipo *java.lang.SecurityException* (ver PBP 1.1:2008) para aplicaciones Ginga-J.

Aplicaciones Ginga-J firmadas, autenticadas y con permisos de acceso a almacenamiento persistente otorgadas deben tener concedidos los privilegios previstos en JAVADTV 1.3:2009, 6.5.

Para aplicaciones con permisos de acceso otorgados, los directorios y subdirectorios a los que tienen acceso, deben crearse automáticamente por la plataforma si no existen. Se recomienda que los directorios y subdirectorios necesarios sean creados luego que se concedan los permisos. El identificador del propietario (*owner*) de los directorios y subdirectorios creados automáticamente por la plataforma debe tener el mismo valor del “*application_id*” (ver ABNT NBR 15606-3:2007, 12.7) de la aplicación en cuestión.

Aplicaciones Ginga-J deben crear archivos o directorios solamente donde tengan permisos de escritura.

Detalles referentes a la liberación de espacio por parte de la plataforma dependen de implementación. Sin embargo, la persistencia de los archivos debe ser garantizada mientras la aplicación esté en ejecución o señalizada en la tabla AIT con un *control code* diferente de KILL o DESTROY.

8.7.5 Acceso a las propiedades del sistema

El acceso a las propiedades del sistema se hará por los métodos *java.lang.System.getProperty, java.lang.System.getProperties* y *java.lang.System.setProperty*.

El uso del método *java.lang.System.setProperties*() no es permitido para aplicaciones Ginga-J.

8.7.6 Soporte a IP sobre canal de interactividad

Para dispositivos en donde sea necesario el establecimiento (*setup*) de conexión, el uso directo de java.net.Socket o java.net.URLConnection debe obligatoriamente resultar en un intento de conexión según los parámetros enviados en el archivo de permisos de la aplicación. Para más informaciones ver JAVADTV 1.3:2009, “*Per Application Policy Schema*”.

Para manipulación de los dispositivos de interactividad disponibles en la plataforma se debe utilizar la API “*Extensive communication device control*” (ver JAVADTV 1.3:2009). Definiciones adicionales sobre el uso de la API estándar para conexión *java.net* y *javax.microedition.io* (ver PBP 1.1:2008) son encontradas en la documentación de la clase *com.sun.dtv.net.NetworkDevice* (ver JAVADTV 1.3:2009), así como en JAVADTV 1.3:2009, 6.2.5.

8.7.7 Filtración de sección MPEG-2

Para obtención de filtros de sección MPEG-2 se debe utilizar la API "*Support MPEG-2 Section Filtering*” (ver JAVADTV 1.3:2009).

8.8 Gestión de aplicaciones

En esta parte de la ABNT NBR 15606, las aplicaciones de televisión digital son denominadas Xlets (ver JAVATV 1.1:2008), teniendo su ciclo de vida gestionado como se describe en 7.1. Estas aplicaciones se ejecutan en un ambiente orientado a servicios, controlado por un Gestor de Aplicaciones (ver JAVADTV 1.3:2009), siendo de responsabilidad de este componente las acciones de cargar, configurar, instar y ejecutar aplicaciones de televisión digital, así como controlar el ciclo de vida y estados de estas aplicaciones. También es de responsabilidad de la gestión de aplicaciones atribuir niveles de prioridad de ejecución a las aplicaciones, además de identificar y minimizar eventuales fallas que acontezcan durante la ejecución de éstas.

Toda esta gestión y monitoreo son realizados internamente por el sistema. A las aplicaciones de televisión debidamente firmadas y certificadas, como se describe en JAVADTV 1.3:2009, 6.2, se permite controlar y/o monitorear el ciclo de vida de estas aplicaciones. Estas funcionalidades están accesibles en la JAVADTV 1.3:2009 a través del paquete:

- com.sun.dtv.application

8.9 Sintonización

La API br.org.sbtvd.net.tuning es una extensión del paquete com.sun.dtv.tuner da JAVADTV 1.3:2009. Las nuevas funciones son *zapping* de canales de forma interactiva y barredura en todas las interfaces de la red existentes en un receptor. Más detalles sobre esas funcionalidades se pueden consultar en el Anexo C.

8.10 Puente NCL

La API de puente NCL, contiene el conjunto de clases disponibles para el puente entre los aplicativos de información y proceso, en ambiente Ginga. Las funciones disponibles en las clases que son descritas abajo permiten el desarrollo de aplicativos de procedimiento Ginga-J incluyendo aplicativos Ginga-NCL, y viceversa. La API de puente NCL Ginga-J controla la presentación de un documento NCL y ofrece recursos para que una aplicación Ginga-J incluida en un documento NCL sea notificada sobre el acaecimiento de eventos de transición realizados sobre el nudo de media que la encapsula.

Si el Java es instado a partir del NCL, la gestión de aplicaciones debe estar de acuerdo con la ABNT NBR 15606-2:2007, 8.5. Si el NCL es instado a partir del JAVA, la gestión de las aplicaciones debe estar de acuerdo con lo descrito en 8.8.

Informaciones complementarias se pueden obtener en la ABNT NBR 15606-2:2007, 11.2 y 10.3.4.3. y Anexo D

8.11 Propiedades de la plataforma

8.11.1 Propiedades de sistema

Las aplicaciones Ginga-J deben tener acceso a las propiedades de plataforma que indiquen características generales, como perfiles y/o funcionalidades soportadas. Luego además de las propiedades ya especificadas en el método *java.lang.System.getProperties* (ver PBP 1.1:2008) y en Ginga-NCL (ver ABNT NBR 15602:2007, 7.2.4), las propiedades listadas en la Tabla 8 deben estar disponibles por medio de los métodos *getProperty*() y *getProperties*() de la clase *java.lang.System* (ver PBP 1.1:2008).

Las propiedades de sistema están descritas en la Tabla 8 y no pueden ser alteradas por las Aplicaciones Ginga-J.

Tabla 8 – Propiedades de sistema

Nombre	Tipo de datos	Descripción
com.sun.dtv.version	Alfanumérico	El número de versión de la especificación JavaDTV implementada por la plataforma (ver JAVADTV 1.3:2009)
com.sun.dtv.net.default.timeout	Numérico	Tiempo máximo de espera (timeout) para establecimiento de conexión
system.gingaj_version	Alfanumérico	El número de versión de la especificación Ginga-J implementada por la plataforma
system.internet_access	Booleano	Indica si el perfil de acceso a la internet esta disponible (valores posibles: "*true*" o "*false*")
system.gingaj_profile	Alfanumérico	El número del perfil de la especificación Ginga-J suportado por la plataforma

8.11.2 Propiedades de usuario

En adición al conjunto de propiedades de sistema, las aplicaciones Ginga-J pueden mantener propiedades específicas por usuario de la plataforma. Para ello, se debe utilizar la API "*User preferences*" (ver JAVADTV 1.3:2009).

La API "*User preferences*" (ver JAVADTV 1.3:2009) también se debe usar para la recepción de notificaciones de cambios en cualesquiera da las propiedades de sistema definidas en 8.11.1.

8.12 Canal de interactividad

El sistema Ginga posee la habilidad de identificar la existencia de dispositivos de canal de interactividad en el receptor, pudiendo establecer la conexión en caso de que éste esté desconectado. La comunicación de datos tras la conexión establecida acontece a través de la API *java.net.**.

8.13 Lista de paquetes mínimos del Ginga-J

8.13.1 Paquetes de la plataforma Java

Los siguientes paquetes (ver CDC 1.1:2008; FP 1.1:2008; PBP 1.1:2008) son incluidos por esta parte de la ABNT NBR 15606:

- java.awt
- java.awt.color
- java.awt.event
- java.awt.font
- java.awt.im
- java.awt.image

- java.beans
- java.io
- java.lang
- java.lang.ref
- java.lang.reflect
- java.math
- java.net
- java.rmi
- java.rmi.registry
- java.security
- java.security.acl
- java.security.cert
- java.security.interfaces
- java.security.spec
- java.text
- java.util
- java.util.jar
- java.util.zip
- javax.microedition.io
- javax.microedition.pki
- javax.microedition.xlet
- javax.microedition.xlet.ixc
- javax.security.auth.x500

8.13.2 Paquetes de la especificación JavaTV 1.1 y JMF 1.0

Los siguientes paquetes (ver JAVATV 1.1:2008) son incluidos por esta parte de la ABNT NBR 15606:

- javax.media
- javax.media.protocol

- javax.tv.graphics
- javax.tv.locator
- javax.tv.media
- javax.tv.net
- javax.tv.service
- javax.tv.service.guide
- javax.tv.service.navigation
- javax.tv.service.selection
- javax.tv.service.transport
- javax.tv.util
- javax.tv.xlet

8.13.3 Paquetes de la especificación JAVADTV 1.3

Los siguientes paquetes (ver JAVADTV 1.3:2009) son incluidos por esta parte de la ABNT NBR 15606:

- com.sun.dtv.application
- com.sun.dtv.broadcast
- com.sun.dtv.broadcast.event
- com.sun.dtv.filtering
- com.sun.dtv.io
- com.sun.dtv.locator
- com.sun.dtv.lwuit
- com.sun.dtv.lwuit.animations
- com.sun.dtv.lwuit.events
- com.sun.dtv.lwuit.geom
- com.sun.dtv.lwuit.layouts
- com.sun.dtv.lwuit.list
- com.sun.dtv.lwuit.painter
- com.sun.dtv.lwuit.plaf

- com.sun.dtv.lwuit.util
- com.sun.dtv.media
- com.sun.dtv.media.audio
- com.sun.dtv.media.control
- com.sun.dtv.media.dripfeed
- com.sun.dtv.media.format
- com.sun.dtv.media.language
- com.sun.dtv.media.text
- com.sun.dtv.media.timeline
- com.sun.dtv.net
- com.sun.dtv.platform
- com.sun.dtv.resources
- com.sun.dtv.security
- com.sun.dtv.service
- com.sun.dtv.smartcard
- com.sun.dtv.test
- com.sun.dtv.transport
- com.sun.dtv.tuner
- com.sun.dtv.ui
- com.sun.dtv.ui.event

8.13.4 Paquetes de la especificación JSSE 1.0.1

Los siguientes paquetes (ver JSSE 1.0.1) son incluidos por esta parte de la ABNT NBR 15606:

- com.sun.net.ssl
- javax.net
- javax.net.ssl
- javax.security.cert

8.13.5 Paquetes de la especificación JCE 1.0.1

Los siguientes paquetes (ver JCE 1.0.1) son incluidos por esta parte de la ABNT NBR 15606:

- javax.crypto
- javax.crypto.interfaces
- javax.crypto.spec

8.13.6 Paquetes de la especificación SATSA 1.0.1

El siguiente paquete (ver SATSA 1.0.1) es incluido por esta parte de la ABNT NBR 15606:

- javax.microedition.apdu

8.13.7 Paquetes específicos Ginga-J

Los siguientes paquetes específicos del GINGA-J son incluidos por esta parte de la ABNT NBR 15606:

- br.org.sbtvd.net
- br.org.sbtvd.net.si
- br.org.sbtvd.net.tuning
- br.org.sbtvd.bridge
- br.org.sbtvd.ui

Los requisitos mínimos para un receptor compatible con Ginga deben estar de acuerdo con la ABNT NBR 15604:2007.

Anexo A
(normativo)

Especificación JavaDTV 1.3

A.1 Consideraciones generales

La especificación JAVADTV 1.3:2009 está compuesta por las API Java DTV adicionadas a la base de los componentes del ambiente de ejecución Java, incluyendo además *Connected Device Configuration, Foundation Profile* y *Personal Basis Profile*.

Java DTV es una especificación que ofrece funcionalidades de un receptor de televisión digital para el desarrollo de aplicaciones basadas en Java. En este Anexo se describen los principales elementos de la especificación JAVADTV 1.3:2009.

A.2 API Java DTV

NOTA Para más detalles sobre la API Java DTV, consultar http://java.sun.com/javarne/technology/javatv/index.jsp

A.2.1 Paquete com.sun.dtv.broadcast

El paquete com.sun.dtv.broadcast permite el acceso a los archivos de *broadcast* y *streams* y se debe implementar de acuerdo con la JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.1.

Tabla A.1 – Clases del paquete com.sun.dtv.broadcast

Clase	Descripción
Clase BroadcastFile	Representa datos del archivo o directorio obtenido a partir del canal de difusión
Clase BroadcastFileEvent	Indica notificaciones de cambio para los datos del BroadcastFile
Clase BroadcastFilesystem	Representa instancias del sistema de archivo obtenido del canal de difusión y montado dentro del sistema de archivos local
Interfaz BroadcastFileListener	Implementada por clases de aplicación que deseen recibir notificaciones de cambio para los datos de BroadcastFile
Clase BroadcastStream	Representa flujos obtenidos a través de archivos obtenidos del canal de difusión
Clase BroadcastException	Todas las excepciones relacionadas con broadcast deben usar esta subclase (que a su vez es una extensión de java.io.IOException) con la intención de hacer más fácil la identificación de los motivos que llevaron a causar la excepción

A.2.2 Paquete com.sun.dtv.smartcard

El paquete com.sun.dtv.smartcard da soporte a funcionalidades adicionales para utilización de smartcards.

La implementación de este paquete tiene que estar de acuerdo con la JAVADTV 1.3:2009, incluyendo las interfaces y clases descritas en la Tabla A.2.

Tabla A.2 – Clases del paquete com.sun.dtv.smartcard

Clases	Descripción
Interfaz CardTerminalListener	Interfaz oyente, responsable por recibir eventos procedentes de lectores de smartcards del dispositivo
Interfaz PassThroughAPDUConnection	Interfaz de marcación que identifica los objetos de conexión específicos de la APDU (*Application Protocol Data Unit*), que realizan comunicación directa con el smartcard
Clase TerminalFactory	Da acceso al(a los) dispositivo(s) lector(es) de smartcards implementados o conectados al dispositivo
Clase CardTerminalEvent	Define un objeto de evento que informa a los *listeners* sobre las cambios de estado acontecidos en el lector de smartcard
Clase CardTerminal	Encapsula las funcionalidades de la parte física del lector de smartcard

A.2.3 Paquete com.sun.dtv.lwuit.events

El paquete *com.sun.dtv.lwuit.events* implementa el estándar de proyecto *Observable* (también utilizado en la API AWT 1.1) que define una arquitectura de tratamiento para lanzamiento y tratamiento de eventos en interfaces gráficas. Todos los eventos son disparados por un *thread* llamado *Event Dispatch Thread* (EDT). Consultar la documentación de la API para más detalles.

Este paquete se debe implementar de acuerdo con la JAVADTV 1.3:2009, en donde deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.3.

Tabla A.3 – Clases del paquete com.sun.dtv.lwuit.events

Clases	Descripción
Interfaz DataChangedListener	Interfaz para tratamiento de evento de *callback*, que son invocados cuando acontecen cambios de estado e indican que la visualización se debe actualizar
Interfaz FocusListener	Oyente de eventos de cambio de enfoque para *Form*, permite que sean atribuidas funcionalidades de acuerdo con el enfoque actual del componente

Tabla A.3 (continuación)

Clases	Descripción
Interfaz SelectionListener	Invocado para indicar un cambio en la selección de la lista modelo
Interfaz StyleListener	Invocado para indicar cambios en una propiedad
Clase ActionEvent	Evento de objeto disparado cuando es invocado un *callback*
Interfaz ActionListener	Interfaz de eventos de *callback* invocado cuando acontece una acción de componente

A.2.4 Paquete com.sun.dtv.filtering

El paquete *com.sun.dtv.filtering* ofrece soporte para acceso a secciones de flujo de transporte MPEG-2. Además de permitir diversos tipos de filtrado, la API utiliza un modelo asíncrono que notifica sobre eventos o errores en la lectura de las secciones.

Este paquete se debe implementar de acuerdo con la JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.4.

Tabla A.4 – Clases del paquete com.sun.dtv.filtering

Clases	Descripción
Clase DataSectionFilterCollection	Esta clase representa una colección de filtros de secciones de datos, en donde éstos se pueden activar y desactivar con una única operación
Clase FilterTimedOutEvent	Este evento es disparado si operaciones de filtros de sección de datos caducan de acuerdo con el período especificado por el método *setTimeOut*()
Clase DataSectionFilterException	Clase base para el lanzamiento de excepciones de la API de filtros de sección de datos
Clase DataSectionAvailableEvent	Este evento indica que una sección de datos fue completamente procesada
Clase DataSectionFilter	Esta es la clase raíz, de todas las clases de filtro
Clase DataSection	Esta clase encapsula una sesión de datos de un flujo de transporte. Los objetos de esta clase también son clonables
Clase IncompatibleSourceException	Clase que informa cuando se abastece un flujo de origen incompatible
Clase DataSectionDataBuffer	Esta clase encapsula una parte de la sesión del flujo de transporte cargado

Tabla A.4 (continuación)

Clases	Descripción
Clase FilteringStoppedEvent	Esta clase se utiliza para informar el final de una operación de filtrado, con una excepción: No informa cuando una *SimpleSectionFilter* termina en condiciones normales (por ejemplo, después que una sección de filtrado ha sido realizada con éxito)
Clase DataUnavailableException	Informa cuando las informaciones de un objeto DataSection no están disponibles
Clase DataSectionFilterEvent	Es una clase base para eventos de la API de filtros de sección
Clase FilterInterruptException	Señaliza que un filtro fue interrumpido antes que todos los datos hayan sido filtrados
Clase DisconnectedException	Indica que cuando el *DataSectionFilterCollection* perdió la conexión por causa de un fallo en la llamada del método *startFiltering*()
Clase InvalidFilterException	Señala que un filtro ha sido definido incorrectamente
Clase FilterResourceUnavailableException	Informa que los requisitos necesarios para una operación de filtrado no se cumplieron
Clase CircularFilter	Esta clase define un filtro de sección destinado para captar flujos continuos de datos sin que sea necesario reiniciar el filtro continuamente
Interfaz FilterEventListener	Esta interfaz de oyente puede ser implementada por clases que exigen eventos de filtrado
Clase ListFilter	Esta clase define un filtro de sección que podrá procesar un conjunto completo de segmentos de una sección de datos que representan una tabla de sección sencilla
Clase SingleFilter	Esta clase define un filtro de sección destinado a capturar una única sección de datos

A.2.5 Paquete com.sun.dtv.ui.event

El subpaquete *com.sun.dtv.ui.event* tiene la función de tratamiento de eventos de interfaz gráfica con el usuario específica para televisión digital.

Este paquete debe ser implementado de acuerdo con la JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.5.

Tabla A.5 – Clases del paquete com.sun.ui.event

Clases	Descripción
Interfaz *KeyListener*	Funciona apenas como un encapsulador para introducir *java.awt.event.KeyListener* en la API LWUIT, ya que ésta no soporta objetos de eventos de teclas y suministra sólo un *ActionListener*
Clase *PlaneSetupEvent*	Evento enviado para todos los registrados cuando una configuración de Plane es alterada
Clase *MouseEvent*	Extiende *java.awt.event.MouseEvent* implementando la interfaz *com.sun.dtv.ui.eventUserInputEvent*. Funciona sólo como una interfaz de marcación, ya que UserInputEvent es derivada de la interfaz *ScarceResource*
Interfaz *UserInputEvent*	Abstracción para eventos de entrada de usuario enviados a todos los consumidores registrados cuando acontece una nueva entrada de usuario a través de un *UserInputDevice*
Interfaz *UserInputEventListener*	Se debe implementar por clases que deseen recibir *UserInputEvent*
Clase *KeyEvent*	Extiende *java.awt.event.KeyEvent* implementando la interfaz *com.sun.dtv.ui.eventUserInputEvent*. Funciona sólo como una interfaz de marcación, ya que *UserInputEvent* es derivada de la interfaz *ScarceResource*
Interfaz *MouseListener*	Funciona solamente como un encapsulador para introducir *java.awt.event.MouseListener* en la API LWUIT, ya que ésta no soporta objetos de eventos de mouse y provee apenas un *ActionListener*
Clase *UserInputEventManager*	Las instancias de esta clase son gestores de eventos que tratan eventos de entrada del usuario generados por componentes gráficos
Clase *RemoteControlEvent*	Extiende *com.sun.dtv.ui.event.KeyEvent* adicionando códigos de tecla específicos de televisión
Interfaz *PlaneSetupListener*	Utilizada para monitorear cambios en la configuración de un *Plane*

A.2.6 Paquete com.sun.dtv.lwuit.plaf

El paquete *com.sun.dtv.lwuit.plaf* permite que la apariencia de la aplicación sea personalizada. En este caso, se utiliza una abstracción de una capa de renderización que puede ser acoplada y configurada (con temas, por ejemplo) en tiempo de ejecución.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.6.

Tabla A.6 – Clases del paquete com.sun.lwui.plaf

Clases	Descripción
Clase *LookAndFeel*	Permite a los programadores personalizar completamente la apariencia de la aplicación a través de métodos adecuados de superposición de imágenes y dimensionado
Clase *UIManager*	El punto central para la gestión de la apariencia de la aplicación permite personalizar los estilos (temas), así como la apariencia
Clase *Style*	Representa la apariencia de un determinado componente
Clase *Border*	Clase base que permite crear un marco para un componente; el marco es dibujado antes del componente rellenando su región marginal
Clase *DefaultLookAndFeel*	Utilizado para renderizar la apariencia estándar del LWUIT 1.1:2008

A.2.7 Paquete com.sun.dtv.media.timeline

El paquete *com.sun.dtv.media.timeline* permite obtener y definir los tiempos de las medias que se notificarán sobre eventos temporales lanzados durante la reproducción de las medias.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Se deben emplear las interfaces y clases de este paquete, descritas de la Tabla A.7.

Tabla A.7 – Clases del paquete com.sun.dtv.media.timeline

Clases	Descripción
Interfaz *MediaTimeListener*	Interfaz de escucha para la recepción de eventos temporales de medias
Interfaz *MediaTimePositionControl*	Controla la definición y obtención de *mediaTime* de un flujo de media. La duración máxima de la media se puede obtener a partir de la interfaz Duration si fuera implementada por el tipo de media en reproducción
Interfaz *MediaTimeEvent*	Evento que informa la aplicación sobre cambios en el tiempo de las medias

A.2.8 Paquete com.sun.dtv.media.language

El paquete *com.sun.dtv.media.language* provee las funcionalidades básicas de consulta y definición de los idiomas de: audio, subtítulos y *Close Captioning*. Los idiomas son codificados en tres lenguas de acuerdo con la ISO 639-2 y ABNT NBR 15603-2 para el uso en descriptores de servicio.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.8.

Tabla A.8 – Clases del paquete com.sun.dtv.media.language

Clases	Descripción
Clase *LanguageNotSupportedException*	Excepción levantada cuando un idioma solicitado no es soportado
Interfaz *LanguageControl*	Control para consulta de los idiomas soportados y para definir un idioma específico para un *player*

A.2.9 Paquete com.sun.dtv.application

El paquete *com.sun.dtv.application* define el modelo de aplicaciones JavaDTV, su ciclo de vida y gestión. Las aplicaciones JavaDTV ejecutan a través de un ambiente de ejecución multitarea Java DTV y es a través de este paquete que el desarrollador puede accesar este ambiente para verificar cuáles son las aplicaciones que están disponibles (*AppManager*), los atributos de cada aplicación (*Application*) y realizar el monitoreo y el control sobre ellas (*ApplicationProxy*). Este paquete también implementa el ciclo de vida de aplicaciones a través de la utilización de clases de la API JavaTV.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben incluirse las siguientes interfaces y clases de este paquete, descritas en la Tabla A.9.

Tabla A.9 – Clases del paquete com.sun.dtv.application

Clases	Descripción
Interfaz *AppManagerListener*	Oyente que notifica sobre alteraciones en los aplicativos disponibles
Clase *AppManager*	Provee control y acceso a las aplicaciones
Interfaz *ApplicationProxy*	Provee control a una aplicación a través del gestor de aplicación
Interfaz *Application*	Contiene los atributos de una aplicación
Clase *AppFilter*	*AppFilter* es llamado para permitir o no una aplicación en un filtro
Class *AppManagerPermission*	Necesaria para las consultas o controles de las aplicaciones
Interfaz *AppProxyListener*	Oyente que recibe notificaciones de alteraciones de estado de la aplicación

A.2.10 Paquete com.sun.dtv.media.audio

El paquete *com.sun.dtv.media.audio* ofrece funcionalidades referentes al control del idioma del audio, como, por ejemplo:

— consultar los idiomas que se pueden seleccionar para su asociación con un Service Component;

— seleccionar el idioma asociado a un Service Component;

— permitir que las aplicaciones reciban notificaciones cuando el idioma de un Service Component sea alterado.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.10.

Tabla A.10 – Clases del paquete com.sun.dtv.media.auto

Clases	Descripción
Interfaz *AudioEvent*	Indica cambios relacionados con la selección del idioma del audio
Interfaz *AudioControl*	Provee control de audio para registrar eventos específicos de audio
Interfaz *AudioListener*	Oyente para cambio en el audio

A.2.11 Paquete com.sun.dtv.test

El paquete *com.sun.dtv.test* provee un *framework* para la realización de pruebas de conformidad. Como normalmente es encontrado en ambientes de prueba Java, el ambiente definido para uso del paquete comprende un servidor y un cliente de pruebas que puede utilizar cualquier medio de comunicación.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.11.

Tabla A.11 – Clases del paquete com.sun.dtv.test

Clases	Descripción
Interfaz *TestCase*	Define los métodos necesarios que deben ser implementados por las pruebas para que se puedan ejecutar en el framework de prueba
Clase *TestHarness*	Provee un punto de entrada y un conjunto de herramientas para los casos de pruebas
Clase *Report*	Agrega el resultado de una prueba: el código, a referencia para la prueba y los motivos relacionados

A.2.12 Paquete com.sun.dtv.tuner

El paquete *com.sun.dtv.tuner provee* API para el acceso y el control de una interfaz de red de difusión (o "*tuner*") utilizados para la recepción de flujos de transporte.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.12.

Tabla A.12 – Clases del paquete com.sun.dtv.tuner

Clases	Descripción
Clase *Tuner*	Representa una interfaz de red o "*tuner*" para la recepción de flujos de transporte de radiodifusión
Clase *TuningCompletedEvent*	Evento que indica la conclusión exitosa de una operación de sintonización

Tabla A.12 (continuación)

Clases	Descripción
Clase *TuningFailedEvent*	Evento que indica la conclusión fracasada de una operación de sintonización
Clase *TuningException*	Excepción que indica informes síncronos de fallas de sintonización
Clase *TuningEvent*	Clase base para los eventos generados por las operaciones de sintonización
Interfaz *TunerListener*	Oyente que recibe eventos de sintonización venidos del *Tuner*
Clase *TuningInitiatedEvent*	Indica el inicio de una operación de sintonización

A.2.13 Paquete com.sun.dtv.lwuit.layouts

El paquete *com.sun.dtv.lwuit.layouts* contiene el modelo de gestión de layout de la LWUIT 1.1:2008, que es similar al modelo utilizado por las APIS AWT/Swing. En ese caso, los gestores de *layout* permiten que un conteiner arregle sus componentes por un conjunto de reglas predefinidas y que puedan adaptarse para tamaño de fuentes o pantalla específicas.

Este paquete se debe implementar en conformidad con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.13.

Tabla A.13 – Clases del paquete com.sun.dtv.lwuit.layouts

Clases	Descripción
Clase *GridLayout*	A través del uso de esta clase como gestor de layout, los componentes son dispuestos en una parrilla de tamaño igual, basado en el espacio disponible
Clase *BorderLayout*	Define un conteiner que organiza y redimensiona sus componentes en una disposición basada en cinco regiones: norte, sur, este, oeste y centro
Clase *LayoutStyle*	*LayoutStyle* se usa para determinar cuánto espacio se utilizará entre los componentes
Clase *BoxLayout*	Dispone elementos en una línea o columna, de acuerdo con una orientación de caja
Clase *FlowLayout*	Dispone los componentes en una cola de manera que cuando llega el fin de una línea, éstos pasan para la línea siguiente
Clase *Layout*	Clase abstracta, que se puede usar para organizar componentes en un conteiner utilizando un algoritmo predefinido
Clase *GroupLayout*	Gestor de layout que agrupa jerárquicamente componentes
Clase *CoordinateLayout*	Permite que los componentes basados en posiciones y tamaños absolutos sean adaptados con base en el espacio disponible para el layout

A.2.14 Paquete com.sun.dtv.broadcast.event

Trata los eventos generados por el canal de broadcast. En este caso, la clase *BroadcastEventManager* trata con todos los eventos y repasa para los BroadcastEventListener registrados.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.14.

Tabla A.14 – Clases del paquete com.sun.dtv.broadcast.event

Clases	Descripción
Clase *BroadcastReceivedEvent*	Representa los eventos que se recibieron a través del canal de difusión
Clase *BroadcastEventManager*	Representa los eventos obtenidos a través del sistema de archivos del canal de difusión
Interfaz *BroadcastEventListener*	Implementada por las clases de la aplicación que requieren notificación de recepción de los datos del *BroadcastEvent*

A.2.15 Paquete com.sun.dtv.lwuit.list

El paquete *com.sun.dtv.lwuit.list* relacionado con la manipulación de *List* que emplea el mismo modelo MVC del *Swing*, incluyendo el estándar de proyecto *renderer*. El paquete posee dos interfaces y dos clases. La interfaz *ListCellRenderer* permite la personalización de la apariencia de la *List* y la *ListModel* define la representación de la estructura de datos que la *List* utiliza como fuente de sus informaciones. Las clases son implementaciones default de las interfaces.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.15.

Tabla A.15 – Clases del paquete com.sun.dtv.lwuit.list

Clases	Descripción
Clase *DefaultListCellRenderer*	Implementación estándar del renderizador basado en un rótulo
Interfaz *ListModel*	Representa la estructura de los datos de la lista, permitiendo así que una lista represente cualquier fuente de datos, en potencia, a través de la referencia para diferentes implementaciones de esta interfaz
Interfaz *ListCellRenderer*	Interfaz que define un renderizador de rótulos de una List. En este caso, representa sólo el marcador de cada célula seleccionada en la *List*
Clase *DefaultListModel*	Implementación estándar de la list model basada en un vector de elementos

A.2.16 Paquete com.sun.dtv.ui

El paquete *com.sun.dtv.ui* es el paquete con funcionalidades de interfaz gráfica con el usuario, específicas para televisión digital.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.16.

Tabla A.16 – Clases del paquete com.sun.dtv.ui

Clases	Descripción
Clase *MatteException*	Un *MatteException* se dispara cuando un aplicativo, por algún motivo, no es capaz de realizar una asociación con un elemento gráfico
Clase *DTVContainerPattern*	El *DTVContainerPattern* es un medio para describir las características de un DTVContainer válido
Clase *Device*	Esta clase es la representación del dispositivo de televisión
Clase *Screen*	Esta clase es la representación de la pantalla del dispositivo de televisión
Clase *DefaultTextLayoutManager*	Esta clase provee un mecanismo estándar de renderización de textos
Clase *FontSpecificationException*	Excepción lanzada cuando un intento de especificar características de una fuente n de manera errónea. En este caso, sólo para fuentes no definidas en un archivo
Interfaz *ViewOnlyComponent*	Esta clase representa cualquier tipo de componente no interactivo del sistema. Utiliza también un mecanismo de configuración de su apariencia
Clase *UserInputDevice*	Base para todos los dispositivos de entrada que se pueden usar para controlar la pantalla del dispositivo
Clase *Mouse*	Esta clase representa el mouse que se puede usar para controlar una Screen particular de un *UserInputDevice*
Clase *AlphaComposite*	Implementa todas las reglas de composición de alfa (transparencia)
Clase *DTVContainer*	Un conteiner de nivel alto en la jerarquía de componentes de la API Java DTV que representa un elemento gráfico conteniendo cualquier cosa visible en un determinado Plane. En ese caso, siempre existe sólo un DTVCointaner para cada Plane
Interfaz *TextLayoutManager*	Define las funcionalidades para el layout de los textos y de su exhibición en la pantalla
Clase *Capabilities*	Describe las capacidades de un Plane

Tabla A.16 (continuación)

Clases	Descripción
Clase *SetupException*	Excepción que puede ser lanzada en varias situaciones en donde se hace un intento de realizar un cambio ilegal en la configuración de uno o más Planos de una *Screen*
Clase *SophisticatedTextLayoutManager*	Esta clase provee una *TextLayoutManager* con más funcionalidades de manera que permita un layout más sofisticado
Clase *PlaneSetupPattern*	Esta clase permite un medio de describir las configuraciones de un plano de exhibición, especificando varias propiedades y su importancia para la aplicación
Clase *AnimatedMatte*	Esta clase representa un matte animado con una máscara de imagen dinámica, en donde los valores de los píxeles determinan la transparencia del matte en un determinado tiempo
Clase *RemoteControl*	Esta clase representa un mando a distancia de televisión que se puede utilizar para controlar una pantalla en particular como un *UserInputDevice*
Interfaz *TextOverflowListener*	Notifica si una cadena de caracteres no cabe en un componente durante el intento de renderizarlo
Clase *Keyboard*	Esta clase representa un teclado que se puede utilizar para controlar determinadas pantallas como un *UserInputDevice*
Clase *FontFileException*	Esta excepción será lanzada en un intento de leer un archivo de fuente con un formato inapropiado
Clase *PlaneSetup*	Describe las características de un *Plane*
Clase *DownloadableFont*	Introduce la posibilidad de bajar fuentes
Interfaz *Matte*	Interfaz básica para todas las clases *Matte*
Interfaz *Animated*	Esta interfaz provee métodos para definir y obtener parámetros de una animación
Interfaz *MatteEnabled*	Permite que componentes hagan composición de matte
Clase *StaticMatte*	Esta clase representa *matte* no animados, por ejemplo, *mattes* que no cambian durante un intervalo de tiempo
Clase *Plane*	Representa una salida de video de un dispositivo de televisión

A.2.17 Paquete com.sun.dtv.media.control

El paquete *com.sun.dtv.media.control* dispone de controles adicionales para la obtención de informaciones sobre la media en exhibición.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.17.

Tabla A.17 – Clases del paquete com.sun.dtv.media.control

Clases	Descripción
Interfaz *FrameRateControl*	Control para obtener la tasa de cuadros
Interfaz *MpegAudioControl*	Control para obtener los parámetros de un flujo de audio MPEG
Interfaz *BitRateControl*	Control para obtener la tasa de bits

A.2.18 Paquete com.sun.dtv.media.dripfeed

El paquete *com.sun.dtv.media.dripfeed* permite la entrega de datos de una imagen estática para un JMF Player, de manera que permita que la aplicación tenga el control sobre los datos. Las imágenes pueden ser de *frames* retirados de una fuente de vídeo.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.18.

Tabla A.18 – Clases del paquete com.sun.dtv.media.dripfeed

Clases	Descripción
Interfaz *DripFeedControl*	Permite la alimentación progresiva de partes de un video en un *Player*
Clase *DripFeedPermission*	Representa los permisos para ingresar al *DripFeedControl*

A.2.19 Paquete com.sun.dtv.security

El paquete *com.sun.dtv.secutity* incluye funcionalidades adicionales de seguridad. Las características básicas son proveídas por las clases en el *java.security* paquete.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.19.

Tabla A.19 – Clases del paquete com.sun.dtv.security

Clases	Descripción
Interfaz *CallbackHandler*	Una aplicación implementa un *CallbackHandler* y repasa para los servicios de seguridad. Así la aplicación puede interactuar con los servicios de seguridad con el objetivo de recuperar informaciones específicas de autenticación como, por ejemplo, *username* y *password* o para exhibir determinada información, por ejemplo mensajes de error o avisos
Interfaz *Callback*	Implementaciones de esa interfaz son pasadas a un *CallbackHandler* permitiendo que servicios de seguridad puedan interactuar con la aplicación llamada y, así, recuperar informaciones específicas de autenticación como, por ejemplo, *username* y *password* o para exhibir determinada información, por ejemplo, mensajes de error o avisos

Tabla A.19 (continuación)

Clases	Descripción
Clase *AuthProvider*	Esta clase define métodos de *login* y *logout* para un proveedor
Clase *LoginException*	Excepción relacionada a operaciones de *login*
Clase *UnsupportedCallbackException*	Disparado cuando es pasada una llamada de *callback* y no puede ser tratada por el destinatario de la llamada

A.2.20 Paquete com.sun.dtv.lwuit.painter

El paquete *com.sun.dtv.lwuit.painter* contiene clases que extienden funcionalidades de la interfaz *com.sun.dtv.lwuit.painter* y que permite dibujar elementos gráficos arbitrarios en el background de componentes (*com.sun.dtv.lwuit.Component*).

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.20.

Tabla A.20 – Clases del paquete com.sun.dtv.lwuit.painter

Clases	Descripción
Clase *PainterChain*	Permite encadenar varios painters de manera que obtenga un efecto en “capa” y donde cada painter dibuja sólo un elemento
Clase *BackgroundPainter*	Dibuja el fondo de pantalla de un componente basado en su estilo

A.2.21 Paquete com.sun.dtv.locator

El paquete *com.sun.dtv.locator* define todos los *Locators* que se utilizarán en el sistema Java DTV.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.21.

Tabla A.21 – Clases del paquete com.sun.dtv.locator

Clases	Descripción
Clase *EntityLocator*	Locator de entidades en los sectores de los transportes de flujos
Clase *URLLocator*	Locator basados en URL
Interfaz *TransportDependentLocator*	Locator que referencia las entidades de un flujo de transporte
Clase *NetworkBoundLocator*	Locator que referencia entidades que están vinculadas a la red.

A.2.22 Paquete com.sun.dtv.resources

El paquete *com.sun.dtv.resources* provee un *framework* básico para dispositivos con recursos limitados.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.22.

Tabla A.22 – Clases del paquete com.sun.dtv.resources

Clases	Descripción
Clase *TimeoutException*	Señala cuando ocurre un *timeout*
Interfaz *ScarceResourceListener*	Notifica sobre la liberación de un determinado recurso escaso
Interfaz *ScarceResource*	Representa recursos que necesitan tratamiento especial para reservar y libertar
Clase *ScarceResourcePermission*	Utilizada para tratar con las diversas autorizaciones relacionadas con recursos escasos
Interfaz *ResourceTypeListener*	Notifica el estatus de alteraciones acontecidas en los recursos del mismo tipo del objeto para el que el oyente se haya conectado.

A.2.23 Paquete com.sun.dtv.net

El paquete *com.sun.dtv.net* extiende el paquete java.net para soporte a control de extensivo de comunicación con dispositivos. En este caso, representa un dispositivo a través de la clase *NetworkDevice.*

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.23.

Tabla A.23 – Clases del paquete com.sun.dtv.net

Clases	Descripción
Interfaz *NetworkDeviceStatusListener*	Oyente para eventos relacionados con dispositivos de red
Clase *NetworkDevicePermission*	Utilizada para tratar con las diversas autorizaciones relacionadas con recursos escasos de dispositivos de red
Clase *NetworkDevice*	Representa cada instancia física de cualquier interfaz de red en soporte el protocolo IP (TCP, UDP). La comunicación puede obtenerse a través de la plataforma

A.2.24 Paquete com.sun.dtv.media.text

El paquete *com.sun.dtv.media.text* permite el acceso al control de subtítulos y "*Closed Captioning*".

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.24.

Tabla A.24 – Clases del paquete com.sun.dtv.media.text

Clases	Descripción
Interfaz *OverlayTextEvent*	Eventos que reportan cambios en: "*OverlayText*", "*Subtitle*" y "*Closed Captioning*"
Interfaz *OverlayTextControl*	Provee control sobre "*Subtitles*" y "*Closed Captioning*"
Interfaz *OverlayTextListener*	Recibe eventos relacionados a "*OverlayText*"

A.2.25 Paquete com.sun.dtv.media.format

El paquete *com.sun.dtv.media.format* es responsable por las configuraciones del formato de video.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.25.

Tabla A.25 – Clases del paquete com.sun.dtv.media.format

Clases	Descripción
Interfaz *VideoFormatControl*	Provee los medios para obtener informaciones sobre el formato y relación de aspecto del video
Interfaz *VideoFormatListener*	Interfaz de escucha que recibe notificación de eventos sobre cambios en la presentación del video
Interfaz *AspectRatioEvent*	La interfaz *VideoFormatEvent* informa sobre cambios habidos en la razón de aspecto
Interfaz *VideoPresentationControl*	Provee los medios para consultar y manipular la presentación del video
Clase *VideoPresentationEvent*	Evento que informa sobre cambios habidos en la presentación del video
Interfaz *ActiveFormatEvent*	El *VideoFormatEvent* informa sobre los cambios en el *Active Format*
Interfaz *VideoPresentationListener*	Relata sobre cambios en la presentación del video, así como todos los tipos de *ContollerEvent*
Interfaz *DecoderFormatEvent*	Evento que informa que el formato del decodificador cambió
Interfaz *ClippingControl*	Control que recupera y define el recorte rectangular del video
Interfaz *VideoFormatEvent*	Evento que informa sobre los cambios en el formato video
Interfaz *BackgroundVideoPresentationControl*	Control de videos mostrados en el fondo de pantalla
Interfaz *ArbitraryVideoScalingControl*	Control para recuperar los factores arbitrarios de escala del video
Clase *Transformation*	Representa un conteiner para informaciones de transformación para un video

A.2.26 Paquete com.sun.dtv.platform

El paquete *com.sun.dtv.platform* provee clases que son específicas de la plataforma Java DTV, en particular las clases relacionadas con el tratamiento de usuarios.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Se deben emplear las interfaces y clases de este paquete, descritas en la Tabla A.26.

Tabla A.26 – Clases del paquete com.sun.dtv.platform

Clases	Descripción
Clase *UserPropertyPermission*	Describe permisos para las propiedades del usuario
Interfaz *UserPropertyListener*	Como alternativa, una aplicación puede adjuntar un *UserPropertyListener* en las propiedades del usuario del subsistema, con la intención de ser notificado sobre cualesquiera alteraciones en las propiedades de los usuarios
Clase *User*	Contiene varios campos y métodos que son específicos para cada usuario de la plataforma

A.2.27 Paquete com.sun.dtv.io

El paquete *com.sun.dtv.io* extiende el paquete *java.io*, dando acceso a los derechos y propiedades de los archivos.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.27.

Tabla A.27 – Clases del paquete com.sun.dtv.io

Clases	Descripción
Clase *FileProperties*	Utilizado para asociar propiedades (o metadatos) con un archivo identificado por su “pathname” en un determinado sistema de archivos
Clase *FileAccessRights*	Provee un medio para definir grupos de niveles de derecho de acceso a un archivo o directorio

A.2.28 Paquete com.sun.dtv.lwuit.animations

En el paquete *com.sun.dtv.lwuit.animations*, todos los componentes son animaciones en potencia y pueden ser rodados en tiempo de ejecución; las transiciones entre *Forms* también son tratadas como parte de este paquete. Las *threads* de animación son tratadas de manera uniforme para disminuir la complejidad para la ejecución en dispositivos computacionalmente limitados.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.28.

Tabla A.28 – Clases del paquete com.sun.dtv.lwuit.animations

Clases	Descripción
Clase *Transition*	Representa una transición con animación entre dos *Forms*; esta clase es usada internamente por el DTVContainer para reproducir una animación cuando se desplaza de un *Form* para el siguiente
Clase *CommonTransitions*	Contiene animaciones de transición comunes
Clase *Motion*	Abstracción de la noción de movimiento físico a lo largo del tiempo entre dos puntos representados por valores numéricos
Interfaz *Animation*	Permite que cualquier componente reciba eventos de animación en intervalos de tiempo y sea actualizado

A.2.29 Paquete com.sun.dtv.service

El paquete *com.sun.dtv.service* provee una interfaz para ingresar al banco de datos del SI (*Service Information*).

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.29.

Tabla A.29 – Clases del paquete com.sun.dtv.service

Clases	Descripción
Clase *SIDatabase*	Provee una forma genérica para ingresar al banco de datos del SI que reside en la plataforma

A.2.30 Paquete com.sun.dtv.media

El paquete *com.sun.dtv.media* es para las funcionalidades relevantes y para los controles de congelar y retornar.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.30.

Tabla A.30 – Clases del paquete com.sun.dtv.media

Clases	Descripción
Interfaz FreezeResumeListener	Permite la aplicación ejecutar los eventos de “congelar” y “retornar” su reproducción
Interfaz FreezeResumeEvent	Indica si los eventos de congelar o retomar acontecieron e identifican si provienen de una aplicación o de un usuario
Interfaz FreezeEvent	Indica si una acción de congelamiento aconteció procedente de una aplicación o de un usuario

Tabla A.30 (continuación)

Clases	Descripción
Interfaz *MediaPresentedEvent*	Este evento es generado después que un *javax.media.Player* es transferido para el estado inicial
Clase *FreezeResumeException*	Esta excepción indica que el método de congelamiento o de retomar fracasó
Interfaz *FreezeResumeControl*	Se debe implementar para permitir que la aplicación “congele” el *Player*
Clase *ConditionalAccessException*	Indica que una media sobre el control de un *Player* o *DataSource* está protegida por acceso condicional
Interfaz *ResumeEvent*	Indica que la acción de continuar la reproducción aconteció a partir de una aplicación o de un usuario

A.2.31 Paquete com.sun.dtv.transport

El paquete *com.sun.dtv.transport* provee acceso a las entidades contenidas en un flujo de transporte.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.31.

Tabla A.31 – Clases del paquete com.sun.dtv.transport

Clases	Descripción
Clase *TransportStream*	Representación de un flujo de transportes y sus servicios asociados
Clase *ConditionalAccessDeniedException*	Esta clase lanzada cuando es solicitado el acceso a las informaciones que están codificadas y cuyo acceso no es permitido por el sistema de seguridad
Clase *ElementaryStream*	Representación de un flujo elemental
Clase *Service*	Representación de un servicio contenido en el flujo de transporte

A.2.32 Paquete com.sun.dtv.lwuit.util

El paquete *com.sun.dtv.lwuit.util* provee funcionalidades utilitarias que son específicas de dominio o no se adecuan a ningún otro paquete de la API.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Se deben emplear las interfaces y clases de este paquete, descritas de la tabla A.32.

Tabla A.32 – Clases del paquete com.sun.dtv.lwuit.util

Clases	Descripción
Clase *Log*	Permite que el desarrollador a través de un framework conectable de logging utilice funcionalidades de log utilizando la API de *file* connector
Clase *Resources*	Está relacionada a la carga de recursos (animaciones, imágenes, temas, fuentes etc.) a partir de un archivo binario generado en el proceso de build

A.2.33 Paquete com.sun.dtv.lwuit

El paquete *com.sun.dtv.lwuit* contiene la jerarquía principal de composición de elementos gráficos (*com.sun.dtv.lwuit.Component* y *com.sun.dtv.lwuit.Container*) de la API LWUIT que sigue un modelo idéntico al de las API Swing/AWT. Pero, al contrario del Swing/AWT, no se utiliza un sistema de ventanas de pantalla llena. En este caso, se utiliza un modelo parecido al de la API MIDP, que utiliza una abstracción de *display* en el cual pueden ser dispuestos los elementos gráficos.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Deben emplearse las interfaces y clases de este paquete, descritas en la Tabla A.33.

Tabla A.33 – Clases del paquete com.sun.dtv.lwuit

Clases	Descripción
Clase *MediaComponent*	Permite la inserción y control de contenido de media rica
Clase *StaticAnimation*	Una imagen capaz de animación
Clase *Graphics*	Abstrae la plataforma de contexto gráfico, permitiendo la portabilidad entre dispositivos MIDP y CDC
Clase *Container*	Implementa el estándar de proyecto *Composite* para *com.sun.dtv.lwuit. Component* de un modo que permite arreglar y relacionar componentes utilizando una arquitectura de gestores de layout conectables
Clase *ComboBox*	Elemento gráfico que representa una lista que permite apenas una selección por vez a través de la elección del usuario
Clase *Font*	Una simple abstracción de fuentes de plataforma y de biblioteca que permite el uso de fuentes que no son soportadas por el dispositivo.
Interfaz *Painter*	Esta interfaz puede ser usada para dibujar en componentes de fondo de pantalla.
Clase *RadioButton*	Tipo específico de *com.sun.dtv.lwuit.Button* que mantiene un estado de selección exclusivamente de un *com.sun.dtv.lwuit.ButtonGroup*
Clase *Calendar*	Permite la selección de valores de fecha y hora
Clase *TabbedPane*	Permite al usuario alternar entre un grupo de componentes haciendo clic en una guía con un determinado título y/o icono
Clase *Command*	Acción referente a los “*soft buttons*” y menú del dispositivo, similar a la abstracción de comando del MIDP y acciones del Swing

Tabla A.33 (continuación)

Clases	Descripción
Clase *CheckBox*	Botón que puede ser marcado o desmarcado y al mismo tiempo exhibir su estado para el usuario
Clase *Form*	Componente de alto nivel que es clase base para interfaces gráficas con el usuario de la LWUIT 1.1:2008. El conteiner se divide en 3 (tres) partes: Title (barra de títulos localizada normalmente en la parte superior), ContentPane (espacio central para disposición de elementos entre Title y MenuBar) y MenuBar (barra de menú localizada normalmente en la parte inferior)
Clase *Dialog*	Un tipo de *Form* que ocupa una parte de la pantalla y aparece como una entidad modal para el desarrollador
Clase *Image*	Abstracción de la plataforma que trata imágenes, permitiendo manipularlas como objetos uniformes
Clase *TextField*	Componente para recepción de entrada de texto de usuario que utiliza una API más leve, sin utilizar el soporte nativo para texto del dispositivo
Clase *ButtonGroup*	Esta clase es utilizada para crear un objetivo de múltiple-exclusión para un conjunto de RadioButtons
Clase *Label*	Permite exhibir etiquetas e imágenes con diferentes opciones de alineación, también funciona como clase base para alineación de la disposición de componentes
Clase *Button*	Componente base para otros elementos gráficos que se pueden pulsar
Clase *List*	Un conjunto (lista) de elementos que son creados utilizando una ListCellRenderer y son extraídos a través de la ListModel
Clase *Component*	Clase base para todos los elementos gráficos de la LWUIT 1.1:2008. Utiliza el estándar de proyecto *Composite* de manera semejante a la relación de *Container* y *Component* del AWT
Clase *TextArea*	Componente gráfico que permite entrada de textos con múltiples líneas editables, también permite exhibir y editar el texto
Clase *AWTComponent*	Extiende la clase *com.sun.dtv.lwuit.Component* como una variante especial que delega las acciones de renderización para *java.awt. Component*

A.2.34 Paquete com.sun.dtv.lwuit.geom

Contiene clases relacionadas a la localización geométrica y cálculos de dimensiones de componentes gráficos.

Este paquete debe ser implementado de acuerdo con JAVADTV 1.3:2009. Se deben emplear las interfaces y clases de este paquete, descrita en la Tabla A.34.

Tabla A.34 – Clases del paquete com.sun.dtv.lwuit.geom

Clases	Descripción
Clase *Point*	Representa una localización en el espacio de coordenadas x e y. Su precisión está basada en enteros
Clase *Rectangle*	Representa una posición rectangular con la dimensión basada en largo y ancho, es útil para medir coordenadas dentro de una aplicación
Clase *Dimension*	Clase utilitaria que almacena valores de largo y ancho y representa una dimensión de un componente o elemento gráfico

Anexo B
(normativo)

Especificación de la API de informaciones de servicio dependiente de protocolo

B.1 Consideraciones generales

Este Anexo describe la API de informaciones de servicio dependiente de protocolo del Ginga-J. Esta API se basa en alteraciones en la especificación ARIB STD-B23:2004, Anexo M. Esto se justifica por la adopción del ISDB-T (ver ARIB STD-B31:2007) como base de la ABNT NBR 15601:2007. Parámetros relevantes para las informaciones sobre el servicio de televisión digital son asociados al método de transmisión utilizado, así, las informaciones sobre el servicio brasileño especificadas por la ABNT NBR 15603:2007 son en gran parte compatibles con la especificación ARIB STD-B10.2008. Sin embargo, la ARIB STD-B23:2004 se basa en las API GEM. Esta parte de la ABNT NBR 15606 es compatible con la plataforma Java DTV, lo que hace necesario introducir adaptaciones en la API de SI. Por este motivo, fue definida y especificada una nueva API.

B.2 API información de servicio dependiente de protocolo

B.2.1 Paquete br.org.sbtvd.net

B.2.1.1 Clase *SBTVDLocator*

SBTVDLocator encapsula SBTVDURL en el objeto. Esta clase extiende la clase *com.sun.dtv.locator.EntityLocator* (ver JAVADTV 1.3:2009).

Los métodos públicos de la clase SBTVDLocator son:

- SBTVDLocator(java.lang.String url) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator
- SBTVDLocator(java.lang.String scheme, int onid, int tsid) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en el siguiente formato.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en el siguiente formato.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en el siguiente formato.

- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en el siguiente formato.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en uno de los siguientes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int[] componenttags) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en uno de los siguientes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int[] componenttags, java.lang.String filePath) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en uno de los siguientes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag, int channelid) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en uno de los siguientes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag, java.lang.String modulename) throws javax.tv.locator.InvalidLocatorException
    - Generación del SBTVD Locator basado en uno de los siguientes formatos.
- SBTVDLocator(java.lang.String scheme, int onid, int tsid, int serviceid, int contentid, int eventid, int componenttag, java.lang.String modulename, java.lang.String resourcename)
    - Generación del SBTVD Locator basado en uno de los siguientes formatos.
- int getChannelId ()
    - Recupera el id del canal.
- int[] getComponentTags()
    - Recupera el array de component_tag.
- int getContentId()
    - Recupera el content_id.
- int getEventId()
    - Recupera el event_id.
- java.lang.String getFilePath()
    - Recupera parte del camino del locator's del nombre del archivo.

- java.lang.String getModuleName()
    - To acquire moduleName.
- int getOriginalNetworkId()
    - Recupera el original_network_id.
- java.lang.String getResourceName()
    - Recupera el resourceName.
- java.lang.String getScheme()
    - Recupera el scheme.
- int getServiceId()
    - Recupera service_id.
- int getTransportStreamId()
    - Recupera transport_stream_id.
- java.net.URL getURL()
    - Recupera la SBTVD URL encapsulada en el objeto SBTVDLocator.

B.2.1.2 Clase *SBTVDNetworkBoundLocator*

SBTVDLocator es conectado con la red. Este tipo de objeto identifica, de manera única, cierta entidad incluyendo la distribución del sistema que transmite la entidad. Por ejemplo, si un determinado servicio es transmitido a través de dos tipos de redes, el servicio puede ser identificado como un servicio común en el SBTVDLocator. Sin embargo, cada servicio transmitido tiene un SBTVDNetworkBoundLocator diferente. Esta clase implementa la interfaz com.sun.dtv.locator.TransportDependentLocator (ver JAVADTV 1.3:2009) y extiende la clase *br.org.sbtvd.net.SBTCDLocator*.

Los métodos públicos de la clase SBTVDNetworkBoundLocator son:

- SBTVDNetworkBoundLocator(SBTVDLocator unboundLocator, int networkId)
    - Generación del network bound locator.
- int getNetworkId()
    - Recupera el network_id.

B.2.2 Paquete *br.org.sbtvd.si*

B.2.2.1 Interfaz *DescriptorTag*

La interfaz *DescriptorTag* define las constantes que corresponden a los valores más comunes del descriptor tag.

Las constantes públicas estáticas de la Clase *DescriptorTag* son:

- static short AUDIO_COMPONENT
    - La constante indica el valor del tag del audio component descriptor especificado en la ARIB STD-B10.
- static short BASIC_LOCAL_EVENT
    - La constante indica el valor del tag del basic local event descriptor especificado en la ARIB STD-B10.
- static short BOARD_INFORMATION
    - La constante indica el valor del tag del board information descriptor especificado en la ARIB STD-B10.
- static short BOUQUET_NAME
    - La constante indica el valor del tag del bouquet name descriptor especificado en la ARIB STD-B10.
- static short BROADCASTER_NAME
    - La constante indica el valor del tag del broadcaster name descriptor especificado en la ARIB STD-B10.
- static short CA_CONTRACT_INFO
    - La constante indica el valor del tag del CA contractor information descriptor especificado en la ARIB STD-B10.
- static short CA_EMM_TS
    - La constante indica el valor del tag del CA_EMM_TS descriptor especificado en la ARIB STD-B10.
- static short CA_IDENTIFIER
    - La constante indica el valor del tag del CA identification descriptor especificado en la ARIB STD-B10.
- static short CA_SERVICE
    - La constante indica el valor del tag del CA service descriptor especificado en la ARIB STD-B10.
- static short CABLE_DELIVERY_SYSTEM
    - La constante indica el valor del tag del cable delivery system descriptor especificado en la ARIB STD-B10.
- static short CAROUSEL_COMPATIBLE_COMPOSITE
    - La constante indica el valor del tag del carrusel compatible composite descriptor especificado en la ARIB STD-B10.

- static short COMPONENT
  - La constante indica el valor del tag del component descriptor especificado en la ARIB STD-B10.
- static short COMPONENT_GROUP
  - La constante indica el valor del tag del component group descriptor especificado en la ARIB STD-B10.
- static short CONNECTED_TRANSMISSION
  - La constante indica el valor del tag del connected transmission descriptor especificado en la ARIB STD-B10.
- static short CONTENT
  - La constante indica el valor del tag del content descriptor especificado en la ARIB STD-B10.
- static short CONTENT_AVAILABILITY
  - La constante indica el valor del tag del contents availability descriptor especificado en la ARIB STD-B10.
- static short COUNTRY_AVAILABILITY
  - La constante indica el valor del tag del country receiving availability descriptor especificado en la ARIB STD-B10.
- static short DATA_COMPONENT
  - La constante indica el valor del tag del data component descriptor especificado en la ARIB STD-B10.
- static short DATA_CONTENTS
  - La constante indica el valor del tag del data contents descriptor especificado en la ARIB STD-B10.
- static short DIGITAL_COPY_CONTROL
  - La constante indica el valor del tag del digital copy control descriptor especificado en la ARIB STD-B10.
- static short DOWNLOAD_CONTENT
  - La constante indica el valor del tag del download contents descriptor especificado en la ARIB STD-B10.
- static short EMERGENCY_INFORMATION
  - La constante indica el valor del tag del emergency information descriptor especificado en la ARIB STD-B10.

- static short EVENT_GROUP
    - La constante indica el valor del tag del event group descriptor especificado en la ARIB STD-B10.
- static short EXTENDED_BROADCASTER
    - La constante indica el valor del tag del extended broadcaster descriptor especificado en la ARIB STD-B10.
- static short EXTENDED_EVENT
    - La constante indica el valor del tag del extended event descriptor especificado en la ARIB STD-B10.
- static short HIERARCHICAL_TRANSMISSION
    - La constante indica el valor del tag del hierarchical transmission descriptor especificado en la ARIB STD-B10.
- static short HYPER_LINK
    - La constante indica el valor del tag del hyper link descriptor especificado en la ARIB STD-B10.
- static short LDT_LINKAGE
    - La constante indica el valor del tag del LDT linkage descriptor especificado en la ARIB STD-B10.
- static short LINKAGE
    - La constante indica el valor del tag del linkage descriptor especificado en la ARIB STD-B10.
- static short LOCAL_TIME_OFFSET
    - La constante indica el valor del tag del local time offset descriptor especificado en la ARIB STD-B10.
- static short LOGO_TRANSMISSION
    - La constante indica el valor del tag del logo transmission descriptor especificado en la ARIB STD-B10.
- static short MOSAIC
    - La constante indica el valor del tag del mosaic descriptor especificado en la ARIB STD-B10.
- static short NETWORK_IDENTIFICATION
    - La constante indica el valor del tag del network identification descriptor especificado en la ARIB STD-B10.
- static short NETWORK_NAME
    - La constante indica el valor del tag del network name descriptor especificado en la ARIB STD-B10.

- static short NODE_RELATION
  - La constante indica el valor del tag del node relation descriptor especificado en la ARIB STD-B10.
- static short NVOD_REFERENCE
  - La constante indica el valor del tag del NVOD reference service descriptor especificado en la ARIB STD-B10.
- static short PARENTAL_RATING
  - La constante indica el valor del tag del parental rating descriptor especificado en la ARIB STD-B10.
- static short PARTIAL_RECEPTION
  - La constante indica el valor del tag del partial reception descriptor especificado en la ARIB STD-B10.
- static short PARTIAL_TRANSPORT_STREAM
  - La constante indica el valor del tag del partial transport stream descriptor especificado en la ARIB STD-B10.
- static short PARTIALTS_TIME
  - La constante indica el valor del tag del partial transport stream time descriptor especificado en la ARIB STD-B10.
- static short REFERENCE
  - La constante indica el valor del tag del reference descriptor especificado en la ARIB STD-B10.
- static short SATELLITE_DELIVERY_SYSTEM
  - La constante indica el valor del tag del satellite delivery system descriptor especificado en la ARIB STD-B10.
- static short SERIES
  - La constante indica el valor del tag del series descriptor especificado en la ARIB STD-B10.
- static short SERVICE
  - La constante indica el valor del tag del service descriptor especificado en la ARIB STD-B10.
- static short SERVICE_LIST
  - La constante indica el valor del tag del service list descriptor especificado en la ARIB STD-B10.
- static short SHORT_EVENT
  - La constante indica el valor del tag del short form event descriptor especificado en la ARIB STD-B10.

- static short SHORT_NODE_INFORMATION
    - La constante indica el valor del tag del short form node information descriptor especificado en la ARIB STD-B10.
- static short SI_PARAMETER
    - La constante indica el valor del tag del SI transmission parameter descriptor especificado en la ARIB STD-B10.
- static short SI_PRIME_TS
    - La constante indica el valor del tag del SI prime TS descriptor especificado en la ARIB STD-B10.
- static short STC_REFERENCE
    - La constante indica el valor del tag del STC reference descriptor especificado en la ARIB STD-B10.
- static short STREAM_IDENTIFIER
    - La constante indica el valor del tag del stream identification descriptor especificado en la ARIB STD-B10.
- static short STUFFING
    - La constante indica el valor del tag del stuffing descriptor especificado en la ARIB STD-B10.
- static short SYSTEM_MANAGEMENT
    - La constante indica el valor del tag del system management descriptor especificado en la ARIB STD-B10.
- static short TARGET_AREA
    - La constante indica el valor del tag del target area descriptor especificado en la ARIB STD-B10.
- static short TERRESTRIAL_DELIVERY_SYSTEM
    - La constante indica el valor del tag del terrestrial delivery system descriptor especificado en la ARIB STD-B10.
- static short TIME_SHIFTED_EVENT
    - La constante indica el valor del tag del time shifted event descriptor especificado en la ARIB STD-B10.
- static short TIME_SHIFTED_SERVICE
    - La constante indica el valor del tag del time shifted service descriptor especificado en la ARIB STD-B10.

- static short TS_INFORMATION
    - La constante indica el valor del tag del TS information descriptor especificado en la ARIB STD-B10.
- static short VIDEO_DECODE_CONTROL
    - La constante indica el valor del tag del video decode control descriptor especificado en la ARIB STD-B10.

B.2.2.2 Interfaz *PMTElementaryStream*

La interfaz *PMTElementaryStream* indica el flujo elemental del servicio (canal). En cada servicio, la PMT está presente para describir los flujos elementales del servicio. Esto significa que el objeto montado con la interfaz indica uno de esos flujos elementales. Cada objeto montado con la interfaz *PMTElementaryStream* es identificado por la combinación de original_network_id, transport_stream_id, *service_id y component_tag* (o elementary_PID).

Los métodos públicos de la clase *PMTElementaryStream* son:

- SBTVDLocator getSBTVDLocator ()
    - Recupera el SBTVDLocator que identifica el flujo elemental.
- int getComponentTag ()
    - Recupera el component tag.
- short getElementaryPID ()
    - Recupera el elementary_PID.
- int getOriginalNetworkID ()
    - Recupera el original network ID.
- int getServiceID ()
    - Recupera el service ID.
- byte getStreamType ()
    - Recupera el stream type ID del flujo elemental.
- int getTransportStreamID ()
    - Recupera el transport stream ID.

B.2.2.3 Interfaz *PMTService*

La interfaz *PMTService* indica el servicio específico que se transmitirá por el flujo de transporte. La información es recuperada de la PMT. Cada objeto montado con la interfaz *PMTService* es identificado por la combinación de *original_network_id*, *transport_stream_id, y service_id*.

Los métodos públicos de la clase *PMTService* son:

- SBTVDLocator getSBTVDLocator()
  - Recupera el SBTVDLocator que identifica este servicio.
- int getOriginalNetworkID ()
  - Recupera el original network ID.
- int getPcrPid()
  - Recupera el PID del PCR.
- int getServiceID()
  - Recupera el service ID.
- int getTransportStreamID()
  - Recupera el transport stream ID.
- SIRequest retrievePMTElementaryStreams(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, short[] somePMTDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
  - Recupera de la PMT la información relevante a los flujos elementales que componen este servicio.

B.2.2.4 Interfaz *PMTStreamType*

La interfaz *PMTStreamType* define las constantes que corresponden a los varios tipos de flujo.

Las constantes públicas estáticas de la Clase *PMTStreamType* son:

- static byte DSMCC_DATA_CAROUSEL
  - La constante indica el tipo de flujo del carrusel de datos definido por la ISO/IEC 13818-1.
- static byte INDEPENDENT_PES
  - La constante indica el tipo de flujo de PES independiente definido por la ISO/IEC 13818-1.
- static byte MPEG1_AUDIO
  - La constante indica el tipo de flujo de audio del MPEG1 definido por la ISO/IEC 13818-1.
- static byte MPEG1_VIDEO
  - La constante indica el tipo de flujo de vídeo del MPEG1 definido por la ISO/IEC 13818-1.
- static byte MPEG2_AAC_AUDIO
  - La constante indica el tipo de flujo de audio del MPEG2AAC definido por la ISO/IEC 13818-1.
- static byte MPEG2_AUDIO
  - La constante indica el tipo de flujo de audio del MPEG2 definido por la ISO/IEC 13818-1.

- static byte MPEG2_VIDEO
    - La constante indica el tipo de flujo de vídeo del MPEG2 definido por la ISO/IEC 13818-1.
- static byte MPEG4_VIDEO
    - La constante indica el tipo de flujo de vídeo del MPEG4 definido por la ISO/IEC 13818-1.
- static byte MPEG4_AVC_VIDEO
    - La constante indica el tipo de flujo de video del H.264/MPEG-4 AVC definido por la ISO/IEC 13818-1.

B.2.2.5 Interfaz *SIBouquet*

La interfaz *SIBouquet* indica la subtabla de la *Bouquet Association Table* (BAT) que describe un *bouquet* específico (junto con el SITransportStreamBAT). Cada objeto que monta la interfaz *SIBouquet* es identificado por el identificador *bouquet_id.si*.SIInformation.

Los métodos públicos de la clase *SIBouquet* son:

- int getBouquetID()
    - Recupera el bouquet ID.
- short[] getDescriptorTags()
    - Este método define la semántica adicional para el SIInformation#getDescriptorTags
- java.lang.String getName()
    - Este método retorna el nombre del bouquet a ser descrito en el descriptor de bouquet.
- SBTVDLocator[] getSIServiceLocators()
    - Este método recupera la lista de SBTVDLocators para identificar el servicio que pertenece al servicio.
- SIRequest retrieveDescriptors(short retrieveMode, java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semántica adicional al primer prototipo del SIInformation#retrieveDescriptors
- SIRequest retrieveDescriptors(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semántica adicional al segundo prototipo del SIInformation#retrieveDescriptors
- SIRequest retrieveSIBouquetTransportStreams(short retrieveMode, java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método provee la información relevante para el transport stream al cual pertenece el bouquet.

B.2.2.6 Interfaz *SIBroadcaster*

La interfaz *SIBroadcaster* indica el proveedor específico en el servicio. Las informaciones retornadas por los métodos son adquiridas de la BIT. Cada objeto que implementa la interfaz *SIBroadcaster* es identificado por el broadcaster_id.

Los métodos públicos de la clase *SIBroadcaster* son:

- int getBroadcasterID ()
  - Retorna el ID del proveedor.
- boolean getBroadcastViewProperty ()
  - Retorna el valor de la propiedad broadcaster's display.
- java.lang.String getName ()
  - Retorna el nombre del proveedor que será descrito en el descriptor de proveedor.
- int getOriginalNetworkID()
  - Retorna el original network ID.
- SBTVDLocator[] getSIServiceLocators()
  - Este método recupera la lista de SBTVDLocator para identificar el servicio que pertenece al proveedor.
- SIRequest retrieveOriginalNetworkDescriptors(short retrieveMode, java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
  - Este método recupera todos los descriptores que son transmitidos en el primer lazo de la BIT.
- SIRequest retrieveOriginalNetworkDescriptors(short retrieveMode, java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
  - Este método recupera el conjunto de descriptores que son transmitidos en el primer lazo de la BIT.

B.2.2.7 Interfaz *SIEvent*

La interfaz *SIEvent* indica un programa específico en el servicio. Cada objeto que implementa la interfaz SIEvent es identificado por la combinación del *original_network_id, transport_stream_id, service_id* y *event_id*. Si el valor de retorno del método es adquirido de la forma simple del descriptor de evento y más de un descriptor está presente, se debe usar el algoritmo siguiente. En el caso que el lenguaje retornado por javax.tv.service.SIManager#getPreferredLanguage sea usado en el descriptor simple de evento, el valor es retornado del descriptor. Caso contrario, dependerá del estado del montaje que debe usarse además de los descriptores simples de evento disponibles. Esta interfaz extiende *br.org.sbtvd.si.SIInformation.*

Los métodos públicos de la clase *SIEvent* son:

- SBTVDLocator getSBTVDLocator()
  - Recupera el SBTVDLocator que identifica el programa.
- java.lang.String[] getAudioComponentDescriptions()
  - Este método retorna la descripción en texto del flujo elemental de audio relevante al programa.
- java.lang.String[] getComponentDescriptions()
  - Este método retorna la descripción en texto del flujo elemental relevante al programa.
- byte[] getContentNibbles()
  - Este método retorna el género del programa.
- java.lang.String[] getDataContentDescriptions()
  - Este método retorna la descripción en texto relevante al data broadcasting program.
- long getDuration()
  - Este método recupera la duración del programa.
- int getEventID()
  - Este método recupera el ID del evento.
- SIExEventInformation[] getExEventInformations()
  - Este método retorna informaciones detalladas relevantes al programa.
- boolean getFreeCAMode ()
  - Este método recupera el valor de mezcla del programa.
- byte[] getLevel1ContentNibbles()
  - Este método retorna el primer paso de la clasificación del contenido del programa.
- java.lang.String getName()
  - Este método retorna el nombre del programa.
- int getOriginalNetworkID()
  - Este método recupera el original network ID.
- byte getRunningStatus()
  - Este método recupera el estado de ejecución del programa.

- java.lang.String getSeriesName()
    - Este método retorna el nombre de la serie relevante al programa.
- int getServiceID()
    - Este método recupera el ID del servicio.
- java.lang.String getShortDescription()
    - Este método retorna la descripción del programa.
- java.util.Date getStartTime()
    - Este método recupera la hora de inicio del programa.
- int getTransportStreamID()
    - Este método recupera el transport stream ID.
- byte[] getUserNibbles()
    - Este método retorna el género relevante al programa.
- SIRequest retrieveSIService(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método recupera el SIService que indica el servicio. El servicio es aquél al cual pertenece el programa indicado por SIEvent.

B.2.2.8 Interfaz *SIInformation*

La interfaz *SIInformation* es una colección de funciones comunes a *SIBouquet, SIBroadcaster, SINetwork, SITransportStream, SIService, PMTService, SIEvent, SITime* y *PMTElementaryStream.*

Las constantes públicas estáticas de la Clase *SIInformation* son:

- static short FROM_CACHE_ONLY
    - La constante es utilizada para el parámetro del modo de adquisición de los métodos de adquisición.
- static short FROM_CACHE_OR_STREAM
    - La constante es utilizada para el parámetro del modo de adquisición de los métodos de adquisición.
- static short FROM_STREAM_ONLY
    - La constante es utilizada para el parámetro del modo de adquisición de los métodos de adquisición.

Los métodos públicos de la clase *SIInformation* son:

- boolean fromActual()
    - Si la información contenida en el objeto que implementa esta interfaz ha sido seleccionada de la tabla “actual” o de la tabla que no distingue “actual” o “not actual”, es retornado “true”.
- com.sun.dtv.transport.TransportStream getDataSource()
    - Este método retorna el objeto com.sun.dtv.transport.TransportStream que se ha seleccionado de la información contenida en el objeto que implementa esta interfaz.
- short[] getDescriptorTags()
    - Este método recupera los valores de las tags de todos los descriptores que forman parte de la versión corriente de este objeto.
- SIDatabase getSIDatabase()
    - Este método retorna la raíz de la estructura jerárquica a la cual pertenece el objeto que implementa esta interfaz.
- java.util.Date getUpdateTime()
    - Este método retorna los últimos día y hora de actualización de esta información incluida en el objeto que implementa esta interfaz.
- SIRequest RetrieveDescriptors (short retrieveMode, java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método retorna todos los descriptores en el orden de envío.
- SIRequest RetrieveDescriptors (short retrieveMode, java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método recupera algunos descriptores.

B.2.2.9 Interfaz *SIIterator*

El objeto que implementa la interfaz *SIIterator* puede acceder al contenido a través de la colección de objetos SI. Con el fin de mantener la consistencia de la colección, no son iniciados algunos accesos al flujo, dependiendo del acceso al contenido.

El método público de la clase *SIIterator* es:

- int numberOfRemainingObjects()
    - El número de objetos mantenidos en el iterador.

B.2.2.10 Interfaz SIMonitoringListener

La interfaz *SImonitoringListener* es implementada por la clase de la aplicación con el fin de recibir los cambios de monitorización del objeto SI.

El método público de la clase *SIMonitoringListener* es:

- void postMonitoringEvent(SIMonitoringEvent anEvent)
    - Este método es llamado por la API del SI con el fin de informar al oyente del evento.

B.2.2.11 Interfaz *SIMonitoringType*

La interfaz *SIMonitoringType* define las constantes que corresponden a cada tipo de información SI en el *SIMonitoringEvent*.

Las constantes públicas estáticas de la Clase *SIMonitoringType* son:

- static byte BOUQUET
    - Constante del objeto SIInformation que indica el bouquet.
- static byte BROADCASTER
    - Constante del objeto SIInformation que indica el proveedor.
- static byte NETWORK
    - Constante del objeto SIInformation que indica la red.
- static byte PMT_SERVICE
    - Constante del objeto SIInformation que indica el PMT service.
- static byte PRESENT_FOLLOWING_EVENT
    - Constante del objeto SIInformation que indica la EIT [Present/Following].
- static byte SCHEDULED_EVENT
    - Constante del objeto SIInformation que indica la EIT [Schedule].
- static byte SERVICE
    - Constante del objeto SIInformation que indica el servicio.

B.2.2.12 Interfaz *SINetwork*

La interfaz *SINetwork* indica la subtabla de la *Network Information Table* (NIT) que describe una red específica (junto con la *SITransportStreamNIT*). Cada objeto que implementa la interfaz *SINetwork* es identificado por el *network_id*.

Los métodos públicos de la clase *SINetwork* son:

- short[] getDescriptorTags()
    - Este método define semánticas adicionales al método SIInformation#getDescriptorTags.
- java.lang.String getName()
    - Este método retorna el nombre de la red descrito en el Network Name Descriptor

- int getNetworkID()
    - Recupera el network ID de esta red.
- SIRequest RetrieveDescriptors(shortretrieveMode,java.lang.ObjectappData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semánticas adicionales al primer prototipo del método SIInformation# retrieveDescriptors.
- SIRequest retrieveDescriptors(short retrieveMode, java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método define semánticas adicionales al segundo prototipo del método SIInformation# retrieveDescriptors.
- SIRequest retrieveSITransportStreams(shortretrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método recupera la información sobre el transport stream que será transmitido a través de la red.

B.2.2.13 Interfaz *SIRetrievalListener*

La interfaz *SIRetrievalListener* debe ser implementada para recibir un evento SI.

El método público de la clase *SIRetrievalListener* es:

- void postRetrievalEvent(SIRetrievalEvent event)
    - Este método es llamado a partir de la SI API implementada de manera que notifique la conclusión del pedido del oyente.

B.2.2.14 Interfaz *SIRunningStatus*

La interfaz *SIRunningStatus* define la constante que corresponde al valor del estado de ejecución del servicio y del evento.

Las constantes públicas estáticas de la Clase *SIRunningStatus* son:

- static NOT_RUNNING
    - byte constante, tal como se define en la ARIB STD-B10, indica que el estado es “not running”.
- static PAUSING
    - byte constante, tal como se define en la ARIB STD-B10, indica que el estado es ejecutando “pausing”.
- static RUNNING
    - byte constante, tal como se define en la ARIB STD-B10, indica que el estado es “running”.

- static STARTS_IN_A_FEW_SECONDS
    - byte constante, tal como se define en la ARIB STD-B10, indica que el estado es “listo para iniciarse en algunos segundos”.
- static UNDEFINED
    - byte constante, tal como se define en la ARIB STD-B10, indica que el estado es “indefinido”.

B.2.2.15 Interfaz *SIService*

La interfaz *SIService* indica un servicio específico que se transmite por uno de los flujos de transporte. Las informaciones obtenidas a través del método de esta interfaz son adquiridas a partir de la SDT. Cada objeto montado con la interfaz *SIService* es identificado por la combinación de los siguientes ID:

Original network ID, Transport stream ID, Service ID

Los métodos públicos de la clase SIService son:

- SBTVDLocator getSBTVDLocator()
    - Este método adquiere el SBTVDLocator para identificar este servicio.
- boolean getEITPresentFollowingFlag()
    - Este método retorna el valor de la EIT_present_following_flag.
- boolean getEITScheduleFlag()
    - Este método retorna el valor de la EIT_schedule_flag.
- int getEITUserDefinedFlag()
    - Este método retorna el valor de las EIT_user_defined_flags.
- boolean getFreeCAMode()
    - Este método retorna el valor del free_CA_mode.
- java.lang.String getName()
    - Este método retorna el nombre que indica el servicio incluido en el service descriptor.
- int getOriginalNetworkID()
    - Este método recupera el original network ID.
- java.lang.String getProviderName()
    - Este método retorna el nombre del prestador de servicios incluidos en el service descriptor.
- byte getRunningStatus()
    - Este método adquiere el estado de ejecución de este servicio.

- int getServiceID()
    - Este método adquiere el servicio ID.
- short getSIServiceType()
    - Este método adquiere el tipo de servicio.
- int getTransportStreamID()
    - Este método adquiere el transporte stream ID.
- SIRequest retrieveFollowingSIEvent(shortretrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquiere las informaciones relevantes para el programa siguiente de EIT [Presente / siguiente].
- SIRequest retrievePMTService(shortretrieveMode,java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método PMTService adquiere las informaciones relevantes para este servicio.
- SIRequest retrievePresentSIEvent(shortretrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquiere las informaciones relevantes para el programa presente de EIT [Presente / siguiente].
- SIRequest retrieveScheduledSIEvents(shortretrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, short[] someDescriptorTags, java.util.Date startTime, java.util.Date endTime) throws br.org.sbtvd.si.SIIllegalArgumentException, br.org.sbtvd.si.SIInvalidPeriodException
    - Este método adquiere las informaciones relevantes para el programa previsto en el periodo designado de EIT [Calendario].

B.2.2.16 Interfaz *SIServiceType*

Esta API es responsable del acceso a informaciones relativas a la definición de *ServiceType*.

Las constantes públicas estáticas de la Clase *SIServiceType* son:

- static BOOKMARK_LIST
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es un “marcador lista de datos “.
- static DATA
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es un “servicio de datos”.

- static DATA_EXCLUSIVE_FOR_ACCUMULATION
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es un “servicio de datos exclusivos de acumulación”.
- static DATA_FOR_ACCUMULATION_IN_ADVANCE
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es un “servicio de datos exclusivos para la acumulación de antecedencia”.
- static DIGITAL_AUDIO
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es un “audio digital “.
- static DIGITAL_TELEVISION
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es del tipo “TV digital “.
- static ENGINEERING_DOWNLOAD
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es un “download engineering”.
- static PROMOTION_DATA
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es “promoción datos”.
- static PROMOTION_SOUND
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es “promoción sonido”.
- static PROMOTION_VIDEO
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es “promoción video”.
- static SPECIAL_AUDIO
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es “ audio especial”.
- static SPECIAL_DATA
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es un “dato especial”.
- static SPECIAL_VIDEO
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es “un video especial “.

- static UNKNOWN
    - Constante, tipo short, tal como se define en la ARIB STD-B10, indica que el tipo de servicio es “desconocido”.

B.2.2.17 Interfaz *SITime*

La interfaz *SITime* provee acceso a las informaciones de la *Time and Date Table* (TDT). Sii el objeto indica la TDT, los métodos retrieveDescriptors y getDescriptorTags se comportan como se especifica cuando no hay descripción, ya que la TDT no posee descriptores.

Esta interfaz indica hora y fecha obtenidas de la tabla (TDT). Si el objeto indica TDT, el comportamiento de los retrieveDescriptors y de las getDescriptorTags es semejante al caso en donde no existe cualquier descriptor (porque no se encuentra Descriptor TDT).

El método público de la clase *SITime* es:

- java.util.Date getTime()
    - Este método adquiere el tiempo codificado en la TDT o en la TOT.

B.2.2.18 Interfaz *SITransportStream*

La interfaz *SITransportStream* es la interfaz base usada para indicar las informaciones relevantes para el flujo de transporte.

El método que recupera el flujo del transporte en la clase *SIDatabase* y en la interfaz *SINetwork* retorna el objeto montado con la interfaz *SITransportStreamNIT* referente a la NIT.El método que recupera el flujo de transporte en la interfaz *SIBouquet* retorna el objeto montado con la interfaz *SITransportStreamBAT* referente a la BAT.

Los métodos públicos de la clase *SITransportStream* son:

- SBTVDLocator getSBTVDLocator()
    - Este método adquiere el SBTVDLocator que identifica el flujo de transporte.
- int getOriginalNetworkID()
    - Este método adquiere el original network ID.
- int getTransportStreamID()
    - Este método adquiere transporte stream ID.
- SIRequest retrieveSIServices(short retrieveMode, java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquiere las informaciones relevantes para el servicio que se transmitirá por el flujo.

B.2.2.19 Interfaz *SITransportStreamBAT*

La interfaz *SITransportStreamBAT* indica informaciones del flujo de transporte recuperadas de la BAT. Todos los métodos que accesan descriptores retornan informaciones de descriptores adquiridos de la BAT. El método que recupera el flujo de transporte en la *SIBouquet* retorna un objeto que implementa esta interfaz.

El método público de la clase *SITransportStreamBAT* es:

- int getBouquetID()
    - El método adquiere el ID del bouquet al cual pertenece este transporte stream.

B.2.2.20 Interfaz *SITransportStreamNIT*

La interfaz *SITransportStreamNIT* indica informaciones del flujo de transporte adquiridas de la NIT. Todos los métodos que accesan descriptores retornan informaciones de descriptores adquiridos de la NIT. El método que recupera el flujo de transporte en el *SIDatabase* o *SINetwork* retorna un objeto que implementa esta interfaz.

El método público de la clase *SITransportStreamNIT* es:

- int getNetworkID()
    - Este método adquiere el ID de la red a la que este flujo de transporte pertenece

B.2.2.21 Clase *Descriptor*

La clase *Descriptor* indica los descriptores de la subtabla.

Los métodos públicos de la clase *Descriptor* son:

- byte getByteAt(int index) throws java.lang.indexOutOfBoundsException
    - Este método obtiene el valor de un byte de la sección de datos del descriptor.
- byte[] getContent()
    - Este método adquiere una copia de la sección de datos de lo descrito (bytes que están después del byte que indica el tamaño del descriptor).
- Short getContentLength()
    - Este método retorna la extensión de la sección de datos indicado en el campo “descriptor length”.
- short getTag()
    - Este método adquiere el tag del descriptor.

B.2.2.22 Clase *SIDatabase*

LA clase *SIDatabase* indica la raíz de la estructura jerárquica de las informaciones del SI. Existe un SIDatabase para cada interfaz de red. Así, existe sólo una *SIDatabase*, si existe apenas una interfaz de red.

Las constantes públicas estáticas de la Clase SIDatabase son:

- static int RETRIEVE_ALL_INFORMATIONS
- static int RETRIEVE_CURRENT_SELECTED

Los métodos públicos de la clase *SIDatabase* son:

- void addBouquetMonitoringListener(SIMonitoringListener listener, int bouquetId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicia el acompañamiento de las informaciones de bouquet.
- void addBroadcasterMonitoringListener(SIMonitoringListener listener, int broadcasterId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicia el acompañamiento de las informaciones de broadcaster.
- void addEventPresentFollowingMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicia el acompañamiento de las informaciones de la EIT [Presente / Siguiendo].
- void addEventScheduleMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId, java.util.Date startTime, java.util.Date endTime) throws br.org.sbtvd.si.SIIllegalArgumentException, br.org.sbtvd.si.SIInvalidPeriodException
    - Este método inicia el acompañamiento de las informaciones de la EIT [schedule].
- void addNetworkMonitoringListener(SIMonitoringListener listener, int networkId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicia el acompañamiento de las informaciones de la red.
- void addPMTServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicia el acompañamiento de las informaciones de la PMT relevantes para el servicio.
- void addServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método inicia el acompañamiento de las informaciones de la SDT relevantes para el servicio.
- static SIDatabase[] getSIDatabase()
    - Este método retorna el objeto SIDatabase(para cada interfaz de red).
- void removeBouquetMonitoringListener(SIMonitoringListener listener, int bouquetId) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método borra el registro del oyente del evento de monitoreo de las informaciones del bouquet.

- void removeBroadcasterMonitoringListener(SIMonitoringListener listener,int broadcasterId) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método borra el registro del oyente del evento de monitoreo de las informaciones del broadcaster.

- void removeEventPresentFollowingMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método borra el registro del oyente del evento de monitoreo de las informaciones de la EIT [presente/siguiente].

- void removeEventScheduleMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Correspondiente a la totalidad programada, este método elimina el registro del evento de monitoreo de la EIT [schedule].

- void removeNetworkMonitoringListener(SIMonitoringListener listener, int networkId) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método borra el registro del evento de monitoreo de la red.

- void removePMTServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId, int serviceId) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método borra el registro del oyente de eventos de monitoreo de las informaciones de la PMT relevantes para el servicio.

- void removeServiceMonitoringListener(SIMonitoringListener listener, int originalNetworkId, int transportStreamId) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método borra el registro del oyente de monitoreo de las informaciones relevantes al servicio.

- SIRequest retrieveActualSINetwork(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método adquiere las informaciones relevantes para la red actual.

- SIRequest retrieveActualSIServices(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método adquiere las informaciones relevantes para el servicio.

- SIRequest retrieveActualSITransportStream (short retrieveMode, java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags). throws br.org.sbtvd.si.SIIllegalArgumentException

    - Este método adquiere las informaciones relevantes para el flujo.

- SIRequest retrievePMTElementaryStreams(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, SBTVDLocator sbtvdLocator, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere informaciones sobre el flujo elemental de la PMT relevantes para el servicio componente del flujo de este SIDatabase.

- SIRequest retrievePMTElementaryStreams(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int serviceId, int componentTag, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere informaciones sobre el flujo elemental de la PMT relevantes para el servicio componente del flujo de este SIDatabase.

- SIRequest retrievePMTService(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, SBTVDLocator sbtvdLocator, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere las informaciones de la PMT relevantes para el servicio.

- SIRequest retrievePMTServices(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int serviceId, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere las informaciones del a PMT relevantes para el servicio a partir del flujo de transporte de este SIDatabase.

- SIRequest retrieveSIBouquets(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int bouquetId, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere las informaciones relevantes para el bouquet.

- SIRequestretrieveSIBroadcaster(shortretrieveMode,java.lang.ObjectappData, SIRetrievalListener listener, int originalNetworkId, int broadcasterId, short [] some DescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere las informaciones relevantes para el broadcaster.

- SIRequest retrieveSIBroadcasters(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int originalNetworkId, short [] some DescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere las informaciones relevantes para el broadcaster especificado por la originalNetworkId.

- SIRequest retrieveSINetworks(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, int networkId, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException

  - Este método adquiere las informaciones relevantes para la red.

- SIRequest retrieveSIService(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, SBTVDLocator sbtvdLocator, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquiere las informaciones relevantes para el servicio.
- SIRequest retrieveSIServices(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener,int originalNetworkId,int transportStreamId, int serviceId, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquiere las informaciones relevantes para el servicio.
- SIRequest retrieveSITimeFromTDT(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquiere de la Time Date Table (TDT) las informaciones sobre tiempo.
- SIRequest retrieveSITimeFromTOT(short retrieveMode,java.lang.Object appData, SIRetrievalListener listener, short[] someDescriptorTags) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método adquiere de la Time Offset Table (TOT) las informaciones sobre tiempo.

B.2.2.23 Clase *SIExEventInformation*

La interfaz *SIExEventInformation* indica los ítems de la descripción y el nombre de los detalles de un programa específico. La información es adquirida de los descriptores de evento en el formato extenso.

Los métodos públicos de la clase *SIExEventInformation* son:

- java.lang.String getDescription()
    - Este método adquiere la descripción del ítem.
- java.lang.String getName()
    - Este método adquiere el nombre del ítem.

B.2.2.24 Clase S*ILackOfResourcesEvent*

El evento de la clase *SILackOfResourcesEvent* es notificado cuando los recursos necesarios para adquirir los datos solicitados en una requisición no están disponibles en el SI. Esta clase extiende br.org.sbtvd.si.SIRetrievalEvent

El método público de la clase *SILackOfResourcesEvent* es:

- SILackOfResourcesEvent(java.lang.Object appData, SIRequest request)
    - Este es el método constructor de este evento.

B.2.2.25 Clase SIMonitoringEvent

El objeto de la clase *SIMonitoringEvent* es transmitido para que el objeto oyente notifique la aplicación del cambio de la información monitorizada. Esta clase extiende java.util.EventObject.

Los métodos públicos de la clase *SIMonitoringEvent* son:

- SIMonitoringEvent(SIDatabase source,byte objectType,int networkId,int bouquetId, int originalNetworkId, int transportStreamId, int broadcasterId, int serviceId, java.util.Date startTime, java.util.Date endTime)
    - Constructor of event object
- int getBouquetID()
    - Este método retorna el bouquet ID
- int getBroadcasterID()
    - Este método retorna el broadcaster ID del broadcaster.
- java.util.Date getEndTime()
    - Este método retorna el fin de la programación cuando la información de los eventos es modificada.
- int getNetworkID()
    - Este método retorna la red ID de la red.
- int getOriginalNetworkID()
    - Este método retorna el ID red original del objeto de SIInformation.
- int getServiceID()
    - Este método retorna el ID del servicio del objeto de SIInformation.
- byte getSIInformationType()
    - Este método adquiere el tipo del SIInformation en el cambio de la información.
- java.lang.Object getSource()
    - Este método adquiere la instancia del SIDatabase que se enviará al evento.
- java.util.Date getStartTime()
    - Este método retorna el inicio de la programación cuando la información de los eventos es modificada.
- int getTransportStreamID()
    - Este método retorna el transport stream ID del objeto SIInformation.

B.2.2.26 Clase *SINotInCacheEvent*

Cuando el pedido de la adquisición del SI en la modalidad de FROM_CACHE_ONLY se ejecute y los datos pedidos no existan en la *cache*, este evento es notificado como una respuesta. Esta clase extiende br.org.sbtvd.si.SIRetrievalEvent.

El método público de la clase *SINotInCacheEvent* es:

- SINotInCacheEvent(java.lang.Object appData, SIRequest request)
    - Constructor del evento

B.2.2.27 Clase *SIObjectNotInTableEvent*

El evento de la clase *SIObjectNotInTableEvent* es notificado cuando la tabla SI que tiene la información sobre la localización del objeto requerido es recuperada pero no contiene dicho objeto. Esta clase extiende br.org.sbtvd.si.SIRetrievalEvent.

El método público de la clase *SIObjectNotInTableEvent* es:

- SIObjectNotInTableEvent(java.lang.Object appData, SIRequest request)
    - Constructor del evento.

B.2.2.28 Clase *SIRequest*

La instancia del objeto de la clase *SIRequest* indica el pedido de adquisición de la aplicación. La aplicación puede cancelar el pedido usando este objeto.

Los métodos públicos de la clase *SIRequest* son:

- boolean cancelRequest()
    - Este método cancela el pedido de la adquisición.
- boolean isAvailableInCache()
    - Este método retorna la disponibilidad de la información si está retornado del flujo o de la cache.

B.2.2.29 Clase *SIRequestCancelledEvent*

El evento de la clase *SIRequestCancelledEvent* es lanzado como una respuesta cuando una requisición es cancelada a través del método de SIRequest.cancelRequest. Esta clase extiende br.org.sbtvd.si.SIRetrievalEvent.

El método público de la clase *SIRequestCancelledEvent* es:

- SIRequestCancelledEvent(java.lang.Object appData, SIRequest request)
    - Constructor

B.2.2.30 Clase *SIRetrievalEvent*

La clase *SIRetrievalEvent* es la clase básica para el evento de conclusión del pedido de adquisición del SI. Solamente un evento es retornado para un pedido de adquisición del SI. Esta clase extiende java.util.EventObject.

Los métodos públicos de la clase *SIRetrievalEvent* son:

- SIRetrievalEvent(java.lang.Object appData, SIRequest request)
    - Constructor del evento

- java.lang.Object getAppData()
    - Este método retorna los datos de la aplicación pasados al método de adquisición.
- java.lang.Object getSource()
    - Este método retorna un objeto SIRequest del evento de origen.

B.2.2.31 Clase *SISuccessfulRetrieveEvent*

El evento de la clase *SISuccessfulRetrieveEvent* es enviado como una respuesta cuando el pedido es terminado normalmente. El resultado puede ser adquirido usando el método getResult. Esta clase extiende br.org.sbtvd.si.SIRetrievalEvent.

Los métodos públicos de la clase *SISuccessfulRetrieveEvent* son:

- SISuccessfulRetrieveEvent(java.lang.Object appData, SIRequest request, SIIterator result)
    - Constructor.
- SIIterator getResult()
    - Este método retorna el objeto de SIIterator que incluye los datos pedidos.

B.2.2.32 Clase *SITableNotFoundEvent*

El evento de la clase *SITableNotFoundEvent* es enviado como una respuesta cuando la tabla SI que debe contener la información requerida no ha sido encontrada. Una de las razones puede ser el hecho de ella no estar siendo transmitida en el flujo conectado al SI database.

El método público de la clase *SITableNotFoundEvent* es:

- SITableNotFoundEvent(java.lang.Object appData, SIRequest request)
    - Constructor

B.2.2.33 Clase *SITableUpdatedEvent*

El evento de la clase *SITableUpdateEvent* es lanzado como una respuesta cuando la tabla, que transmite la información sobre el objeto blanco de la requisición del SI es actualizada y la a información del descriptor en conformidad con el objeto antiguo no está disponible. En este caso, la aplicación debe inicialmente actualizar el objeto de SIInformation. Entonces la información sobre el descriptor se debe pedir otra vez. Esta clase extiende br.org.sbtvd.si.SIRetrievalEvent.

El método público de la clase *SITableUpdatedEvent* es:

- SITableUpdatedEvent(java.lang.Object appData, SIRequest request)
    - Constructor Estándar.

B.2.2.34 Clase *SIUtil*

La clase *SIUtil* incluye la función de utilidad relevante al SI.

El método público de la clase *SIUtil* es:

- static java.lang.String convertSIStringToJavaString(byte[] sbtvdSIText, int offset, int length) throws br.org.sbtvd.si.SIIllegalArgumentException
    - Este método convierte el string de texto codificado en el objeto string de Java basado en la ARIB STD-B10:2008 Parte 2, Apéndice A. Este método es heredado de Class java.lang.Object.

B.2.2.35 Clase *SIException*()

La clase *SIException*() es la base de la jerarquía de excepciones de la SI. Esta clase extiende java.lang.Exception

Los métodos públicos de la clase *SIException*() es:

- SIException()
    - Constructor estándar de excepción.
- SIException(String message)
    - Constructor con parámetro (String) para indicar el motivo de la excepción.

B.2.2.36 Clase *SIIllegalArgumentException()*

La clase *SIIllegalArgumentException*() es lanzada cuando es pasado más de un argumento impropio (por ejemplo. valores numéricos fuera del espacio). Esta clase extiende br.sbtvd.si.SIException.

El método público de la clase *SIIllegalArgumentException*() es:

- SIIllegalArgumentException() ()
    - Constructor estándar de excepción.
- SIIllegalArgumentException (String message)
    - Constructor con parámetro (String) para indicar el motivo de la excepción.

B.2.2.37 Clase *SIInvalidPeriodException*

La clase *SIInvalidPeriodException* acontece cuando la extensión de tiempo especificada es impropia, por ejemplo, el tiempo de inicio está más atrasado.

Los métodos públicos de la clase *SIInvalidPeriodException* son:

- SIInvalidPeriodException()
    - Constructor estándar de excepción.
- SIInvalidPeriodException(java.lang.String reason)
    - Constructor de excepción que tiene una razón para ser especificado.

Anexo C
(normativo)

Especificación API de extensión para sintonía – Paquete *br.org.sbtvd.net.tuning*

C.1 Clase *ChannelManager*

La clase *ChannelManager* especifica un objeto responsable por la actividad de *zapping* (cambio) de canales sintonizables a través de la interfaz de red (terrestre, cable, satélite, IPTV) existente en el receptor de televisión digital.

Los métodos públicos son los siguientes:

- static ChannelManager getInstance ()
    - Retorna un objeto (instancia) ChannelManager.
- int getNumberofChannels ()
    - Retorna el número de canales encontrados en todas las interfaces de red disponibles en el sistema.
- Channel [] getChannels ()
    - Retorna el número de canales encontrados en todas las interfaces de red disponibles en el sistema.
- void tuneChannel (Channel ch, com.sun.dtv.tuner.TunerListener lis) throws com.sun.dtv.tuner.TuningException
    - Sintoniza asíncronamente el canal proveído por el parámetro entero num. Este método lanza una excepción del tipo com.sun.dtv.tuner.TuningException en caso de fallo en la sintonía.
- void tuneNextChannel (com.sun.dtv.tuner.TunerListener lis) throws com.sun.dtv.tuner.TuningException
    - Sintoniza asíncronamente el próximo canal de la tabla TransportStream generada durante la barredura. Este método lanza una excepción del tipo com.sun.dtv.tuner.TuningException en caso de fallo en la sintonía.
- void tunePreviousChannel (com.sun.dtv.tuner.TunerListener lis) throws com.sun.dtv.tuner.TuningException
    - Sintoniza de manera asíncrona el canal anterior de la tabla TransportStream generada durante la barredura. Este método lanza una excepción del tipo com.sun.dtv.tuner.TuningException en caso de fallo en la sintonía.

C.2 Clase *Channel*

La clase *Channel* representa un objeto que contiene los datos de los canales detectados durante la barredura. A partir de las informaciones de esta clase, es posible, por ejemplo, sintonizar un canal a través de su número virtual (*remote control key id*).

Los métodos públicos son los siguientes:

- com.sun.dtv.transport.TransportStream getTransportStream ()
    - Retorna el objeto que contiene el canal
- String getNetworkName ()
    - Retorna la descripción de la red en la cual el canal se encuentra en el momento.
- String getTransportStreamName ()
    - Retorna la descripción del flujo de transporte en el cual el canal se encuentra en el momento.
- int getRemoteControlKeyId()
    - Retorna el número virtual del canal

Anexo D
(normativo)

Especificación API de puente NCL

D.1 Consideraciones generales

Los paquetes *br.org.sbtvd.bridge* y *br.org.sbtvd.bridge.ncl* contienen el conjunto de clases disponibles para el puente entre los aplicativos escritos en los lenguajes NCL y Java, en ambiente Ginga. Las funciones disponibles en las clases que son descritas en D.2.1 pueden utilizarse para el desarrollo de aplicativos Ginga-J incluyendo aplicativos Ginga-NCL, así como el desarrollo de aplicaciones Ginga-NCL, incluyendo Xlets Java.

En el primer caso, la API de puente NCL Ginga-J hace posible la presentación y manipulación de un documento NCL en una aplicación Java, a través de la clase NCLPlayer, manteniendo el documento original preservado durante todo el proceso de exhibición. Las clases que añaden tales funcionalidades son disponibilizadas por el paquete *br.org.sbtvd.bridge*.

En el segundo caso, la API de puente viabiliza que un documento NCL también es capaz de incluir Xlets Ginga-J como uno de sus nudos de media (elemento <media>). Un elemento <media> conteniendo un código Java puede definir áncoras (a través de elementos <area>), y atributos (a través de elementos <property>). Las transiciones aplicadas al Xlet invocarán los métodos de la interfaz *javax.microedition.xlet.Xlet* (ver PBP 1.1:2008), que representan las transiciones de su máquina de estados. Las transiciones aplicadas a los nudos y áncoras deben generar eventos de la clase NCLEvent, que encapsulan la transición y el identificador del nudo o áncora en cuestión. En D.2.2 se presentan las clases que son responsables por la comunicación del Ginga-NCL con el ambiente Ginga-J, cuando un documento NCL incluye una aplicación Ginga-J. Dichas clases componen el paquete br.org.sbtvd.bridge.ncl.

Informaciones complementarias se pueden obtener en la ABNT NBR 15606-2:2007, 10.3.4.3 y 11.2.

D.2 API de puente NCL

D.2.1 Paquete br.org.sbtvd.bridge

D.2.1.1 Clase *NCLPlayer*

La clase NCLPlayer es una clase que representa un exhibidor para un documento NCL, siendo un componente gráfico que extiende *java.awt.Component*. El tratamiento delos eventos de entrada (de teclas, por ejemplo) se hará por el exhibidor NCL (los eventos son repasados por el ambiente Ginga-J para el Ginga-NCL) mientras el NCLPlayer tenga el enfoque de interacción de entre los componentes utilizados en el Xlet en cuestión.

Las constantes públicas estáticas de la Clase NCLPlayer son:

- static int PLAYING
    - Identificación para documento en ejecución.

- static int PAUSED
    - Identificación para documento en pausado.
- static int STOPPED
    - Identificación para documento en parado.

Los métodos públicos de la clase NCLPlayer son:

- NCLPlayer(java.net.URL documentURL)
    - Método constructor para NCLPlayer, que recibe como parámetro una instancia de la clase java.net.URL como localizador del documento.
- void addNCLPlayerEventListener(NCLPlayerEventListener listener, long nclPlayerEventPlayerMask)
    - Este método registra un NCLPlayerEventListener para recibir todos los NCLPlayerEvents distribuidos por la máquina asociada al nudo que se vincula al valor long eventMask proveído.
- void removeNCLPlayerEventListener(NCLPlayerEventListener listener)
    - Quita un NCLPlayerEventListener de la clase de recepción de los NCLPlayerEvents distribuidos.
- NCLPlayerEventListener[] getNCLPlayerEventListeners()
    - Retorna una lista de todos los NCLPlayerEventListeners registrados en este NCLPlayer. Los objetos oyentes adicionados varias veces aparecen sólo una vez en la lista retornada.
- NCLPlayerEventListener[] getNCLPlayerEventListeners(long nclPlayerEventMask)
    - Retorna una lista de todos los NCLPlayerEventListeners registrados en este NCLPlayer que oyen a todos los tipos de eventos indicados en el valor long eventMask. Los objetos oyentes adicionados varias veces aparecen sólo una vez en la lista retornada.
- java.net.URL getDocumentURL()
    - Retorna un objeto de la clase java.net.URL que es el localizador del documento NCL siendo manipulado por el objeto NCLPlayer en cuestión.
- java.lang.String getPropertyValue(java.lang.String propertyId)
    - Retorna una instancia de String con el valor de la propiedad definida por el parámetro String propertyId.
- int getStatus()
    - Retorna un valor entero representando el estado del objeto NCLPlayer (PLAYING – en ejecución, PAUSED – pausado, STOPPED – parado).
- void setDocument(java.net.URL documentURL)
    - Define el documento NCL a ser manipulado por el NCLPlayer, recibiendo un objeto de la clase java.net.URL como identificador del documento. El nuevo estado de ejecución del NCLPlayer deberá ser definido como parado (STOPPED).

- boolean startDocument(java.lang.String interfaceId)
    - Comienza la reproducción de un documento NCL empezando la presentación a partir de la interfaz de documento especificada por la instancia de String interfaceId. Retorna true en caso de éxito, y false en caso contrario.
- boolean stopDocument()
    - Para la presentación de un documento NCL. Todos los eventos del documento que están en marcha deben obligatoriamente ser parados. Retorna true en caso de éxito, y false en caso contrario.
- boolean pauseDocument()
    - Pausa la presentación de un documento NCL. Todos los eventos del documento que están en marcha deben obligatoriamente ser pausados. Retorna true en caso de éxito, y false caso contrario.
- boolean resumeDocument()
    - Retoma la presentación de un documento NCL. Todos los eventos de documentos que fueron previamente pausados por el método pauseDocument() deben obligatoriamente ser retomados. Retorna true en caso de éxito, y false en caso contrario.
- NCLEdit getNCLEdit()
    - Retorna una instancia de la clase NCLEdit, que ofrece funcionalidades de edición del documento NCL en tiempo de exhibición.

D.2.1.2 Clase *NCLPlayerEvent*

La clase NCLPlayerEvent extiende java.util.EventObject y representa el evento raíz para todos los eventos NCL generados por la aplicación NCL encapsulada por una instancia de la clase en cuestión. Las máscaras de evento definidas en esta clase se utilizan para especificar cuáles son los tipos de eventos que un NCLPlayerEventListener debe oír. Informaciones adicionales se pueden obtener en la ABNT NBR 15606-2:2007, 10.3.4.3.

Las constantes públicas estáticas de la Clase NCLPlayerEvent son:

- public static final int PRESENTATION_START = 1;
    - Máscara de evento para eventos de presentación (tipo presentation) cuya acción (campo action) fue el inicio de la reproducción (start) de un nudo o áncora.
- public static final int PRESENTATION_STOP = 2;
    - Máscara de evento para eventos de presentación (tipo presentation) cuya acción (campo action) fue la terminación de la reproducción (stop) de un nudo o áncora.
- public static final int PRESENTATION_ABORT = 4;
    - Máscara de evento para eventos de presentación (tipo presentation) cuya acción (campo action) fue abortar la reproducción (abort) de un nudo o áncora.

- public static final int PRESENTATION_PAUSE = 8;
    - Máscara de evento para eventos de presentación (tipo presentation) cuya acción (campo action) fue la pausa de la reproducción (pause) de un nudo o áncora.
- public static final int PRESENTATION_RESUME = 16;
    - Máscara de evento para eventos de presentación (tipo presentation) cuya acción (campo action) fue la retomada de la reproducción (resume) de un nudo o áncora.
- public static final int ATTRIBUTION_SET = 32;
    - Máscara de evento para eventos de atribución (tipo attribution) cuya acción (campo action) fue la definición (set) de un parámetro de un nudo o áncora.
- protected int id
    - Identificación del evento.

Los métodos públicos de la clase NCLPlayerEvent son:

- NCLPlayerEvent(Object source, int id, String value)
    - Método constructor para NCLPlayerEvent, que recibe como parámetro una referencia (source) del objeto que originó el evento, un entero (id) que identifica el evento y un identificador (value) del nudo o áncora relacionado al evento
- int getID()
    - Retorna el tipo de evento.
- String getValue()
    - Retorna el identificador del nudo o áncora relacionado al evento.

D.2.1.3 Interfaz *NCLPlayerEventListener*

La interfaz *Listener* debe ser implementada por quien desea recibir notificación de eventos distribuidos para objetos que son elementos de la clase *NCLEvent*. Extiende la interfaz java.util.EventListener

La aplicación que se interesa en monitorear los eventos NCL de un *NCLPlayer* implementa esta interfaz registrándose con el *NCLPlayer* a través del método *NCLPlayer.addNCLPlayerEventListener()*. Cuando un evento es distribuido en el *NCLPlayer*, es ejecutado el método *eventDispatched* de ese objeto.

El método público de la interfaz *NCLPlayerEventListener* es:

- void NCLPlayerEventDispatched(NCLPlayerEvent event)
    - Método ejecutado cuando un evento es distribuido en el NCLPlayer.

D.2.1.4 Clase *NCLGingaSettingsNode*

La clase *NCLGingaSettingsNode* es la clase que representa un nudo NCL cuyos atributos son variables globales definidas por el autor del documento o variables de ambiente que pueden ser manejadas por el procesamiento del documento NCL. La lista completa de esas variables de ambiente se presenta en la ABNT NBR 15606-2.

Los métodos públicos de la clase *NCLGingaSettingsNode* son:

- NCLGingaSettingsNode(java.lang.String nodeId)
    - Método constructor para NCLGingaSettingsNode, que recibe como parámetro una String identificadora única.
- String getValue(String value)
    - Retorna el valor de la variable de acuerdo con la descripción de la variable de ambiente pasada como String. La lista completa de variables de ambiente disponibles se encuentra en la ABNT NBR 15606-2:2007, Tabla 12.

D.2.1.5 Clase *NCLEdit*

La clase *NCLEdit* ofrece métodos para edición de un documento NCL, que encapsulado en un objeto de la clase *NCLPlayer* instancia objetos de la clase NCLEdit (método *getNCLEdit*) asociados. Comandos de edición procedentes de la instancia de NCLEdit alteran solamente la presentación del documento NCL (representada por el objeto NCLPlayer) – el documento original es preservado durante todo el proceso de edición, conforme especificado para comandos de edición NCL en ABNT NBR 15606-2:2007.

Los métodos públicos de la clase *NCLEdit* son:

- boolean addRegion(java.lang.String regionBaseId, java.lang.String regionId, java.lang.String regionStr)
    - Adiciona un elemento <region> en el documento NCL, como miembro de la región base identificada por la String regionBaseId, como hijo del elemento identificado por la String regionId y con su definición en la String regionStr. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeRegion(java.lang.String regionId)
    - Quita el elemento <region> identificado por la String regionId del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addRegionBase(java.lang.String regionBaseStr)
    - Adiciona el elemento <regionBase> descrito en la String regionBaseStr al elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeRegionBase(java.lang.String regionBaseId)
    - Quita el elemento <regionBase> identificado por la String regionBaseId del documento NCL. Retorna true en caso de éxito, y false en caso contrario.

- boolean addRule(java.lang.String ruleStr)
    - Adiciona un elemento <rule> descrito en la String ruleStr como integrante del elemento <ruleBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeRule(java.lang.String ruleId)
    - Quita el elemento <rule> identificado en la String ruleId del elemento <ruleBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addRuleBase(java.lang.String ruleBaseStr)
    - Adiciona un elemento <ruleBase> descrito en la String ruleBaseStr como integrante del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeRuleBase(java.lang.String ruleBaseId)
    - Quita el elemento <ruleBase> identificado en la String ruleBaseId del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addConnector(java.lang.String connectorStr)
    - Adiciona un elemento <connector> descrito en la String connectorStr como integrante del elemento <connectorBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeConnector(java.lang.String connectorId)
    - Quita el elemento <connector> identificado en la String connectorId del elemento <connectorBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addConnectorBase(java.lang.String connectorBaseStr)
    - Adiciona un elemento <connectorBase> descrito en la String connectorBaseStr como integrante del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeConnectorBase(java.lang.String connectorBaseId)
    - Quita el elemento <connectorBase> identificado en la String connectorBaseId del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addDescriptor(java.lang.String descriptorStr)
    - Adiciona un elemento <descriptor> descrito en la String descriptorStr como integrante del elemento <descriptorBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeDescriptor(java.lang.String descriptorId)
    - Quita el elemento <descriptor> identificado en la String descriptorId del elemento <descriptorBase> do documento NCL. Retorna true en caso de éxito, y false en caso contrario.

- boolean addDescriptorSwitch(java.lang.String descriptorSwitchStr)
    - Adiciona un elemento <descriptorSwitch> descrito en la String descriptorSwitchStr como integrante del elemento <descriptorBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeDescriptorSwitch(java.lang.String descriptorSwitchId)
    - Quita el elemento <descriptorSwitch> identificado en la String descriptorSwitchId del elemento <descriptorBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addDescriptorBase(java.lang.String descriptorBaseStr)
    - Adiciona un elemento <descriptorBase> descrito en la String descriptorBaseStr como integrante del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeDescriptorBase(java.lang.String descriptorBaseId)
    - Quita el elemento <descriptorBase> identificado en la String descriptorBaseId del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addTransition(java.lang.String transitionStr)
    - Adiciona un elemento <transition> descrito en la String transitionStr como integrante del elemento <transitionBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeTransition(java.lang.String transitionId)
    - Quita el elemento <transition> identificado en la String transitionId del elemento <transitionBase> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addTransitionBase(java.lang.String transitionBaseStr)
    - Adiciona un elemento <transitionBase> descrito en la String transitionBaseStr como integrante del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeTransitionBase(java.lang.String transitionBaseId)
    - Quita el elemento <transitionBase> identificado en la String transitionBaseId del elemento <head> del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addImportBase(java.lang.String docBaseId, java.lang.String importBaseStr)
    - Adiciona a un elemento base NCL identificado en la String docBaseId (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> o <connectorBase>) la definición del elemento <importBase> contenida en la String importBaseStr en el documento NCL. Retorna true en caso de éxito, y false en caso contrario.

- boolean removeImportBase(java.lang.String docBaseId, java.lang.String importBaseId)
    - Quita el elemento <importBase> identificado en la String importBaseId de un elemento base NCL identificado en la String docBaseId (<regionBase>, <descriptorBase>, <ruleBase>, <transitionBase> o <connectorBase>) del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean addImportedDocumentBase(java.lang.String importedDocumentBaseStr)
    - Adiciona al elemento <head> del documento NCL la definición del elemento <importedDocumentBase> contenida en la String importedDocumentBaseStr. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeImportedDocumentBase(java.lang.String importedDocumentBaseId)
    - Quita del elemento <head> del documento NCL el elemento <importedDocumentBase> identificado por la String importedDocumentBaseId. Retorna true en caso de éxito, y false en caso contrario.
- boolean addImportNCL (java.lang.String importNCLStr)
    - Adiciona al elemento <importedDocumentBase> del documento NCL la definición del elemento <importNCL> contenida en la String importNCLStr. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeImportNCL(java.lang.String importNCLId)
    - Quita del elemento <importedDocumentBase> del documento NCL el elemento <importNCL> identificado por la String importNCLId. Retorna true en caso de éxito, y false en caso contrario.
- boolean addNode(java.lang.String compositeId, java.lang.String nodeStr)
    - Adiciona a un nudo de composición NCL identificado en la String compositeId (<body>, <context> o <switch>) la definición de un nudo NCL (<media>, <context> o <switch>) contenida en la String nodeStr. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeNode(java.lang.String compositeId, java.lang.String nodeId)
    - Quita de un nudo de composición NCL identificado en la String compositeId (<body>, <context> o <switch>) la definición de un nudo NCL (<media>, <context> o <switch>) identificada en la String nodeId. Retorna true en caso de éxito, y false en caso contrario.
- boolean addInterface(java.lang.String nodeId, java.lang.String interfaceStr)
    - Adiciona una interfaz del NCL (elemento <port>, <area>, <property> o <switchPort>) descrita en la String interfaceStr a un nudo (elemento <media>, <body>, <context> o <switch>) identificado por la String nodeId del documento NCL. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeInterface(java.lang.String nodeId, java.lang.String interfaceId)
    - Quita una interfaz del NCL (elemento <port>, <area>, <property> o <switchPort>) identificado en la String interfaceId de un nudo (elemento <media>, <body>, <context> o <switch>) identificado por la String nodeId del documento NCL. Retorna true en caso de éxito, y false en caso contrario.

- boolean addLink(java.lang.String compositeId, java.lang.String linkStr)
    - Adiciona a un nudo de composición NCL identificado en la String compositeId (<body>, <context> o <switch>) la definición de un elemento <link> NCL contenida en la String linkStr. Retorna true en caso de éxito, y false en caso contrario.
- boolean removeLink(java.lang.String compositeId, java.lang.String linkId)
    - Quita de un nudo de composición NCL identificado en la String compositeId (<body>, <context> o <switch>) la definición de un elemento <link> NCL identificada en la String linkId. Retorna true en caso de éxito, y false en caso contrario.
- boolean setPropertyValue(java.lang.String propertyId, java.land.String value)
    - Atribuye el valor de la String value a una propiedad identificada por propertyId. El valor de la instancia de String propertyId debe obligatoriamente identificar un atributo name de un elemento <property> o un atributo id de elemento <switchPort>. El <property> o <switchPort> debe obligatoriamente pertenecer a un nudo (elemento <body>, <context>, <switch> o <media>) de un documento NCL. Retorna true en caso de éxito, y false en caso contrario.

D.2.2 Paquete br.org.sbtvd.bridge.ncl

D.2.2.1 Clase *NodeManager*

La clase estática NodeManager posee los métodos para registro de NCLEventListeners de forma que un Xlet Ginga-J asociado a un nudo de media NCL (elemento <media>) pueda recibir eventos (encapsulados en instancias de la clase NCLEvent) del ambiente Ginga-NCL. Informaciones adicionales pueden obtenerse en la ABNT NBR 15606-2:2007, 10.3.4.3 y 11.2.

Los métodos públicos de la clase *NodeManager* son:

- static void addNCLEventListener(NCLEventListener listener)
    - Este método registra un NCLEventListener para recibir todas las instancias de NCLEvent distribuidas por el ambiente Ginga-NCL para el Xlet asociado a un nudo de media (elemento <media>) en una aplicación Ginga-NCL.
- static void removeNCLEventListener(NCLEventListener listener)
    - Quita un NCLEventListener previamente registrado.
- static NCLEventListener[] getNCLEventListeners()
    - Retorna una lista de todas las instancias de NCLEventListener registradas en el NodeManager. Los objetos oyentes adicionados varias veces aparecen solamente una vez en la lista retornada.

D.2.2.2 Clase *NCLEvent*

La clase *NCLEvent* extiende java.util.EventObject y representa la clase de evento raíz para todos los eventos generados por el formateador NCL que maneja un documento incluyendo un Xlet Ginga-J. Las máscaras de evento definidas en esta clase son utilizadas para especificar cuáles son los tipos de eventos que un *NCLEventListener* debe oír. Informaciones adicionales se pueden obtener en la ABNT NBR 15606-2:2007, 10.3.4.3 y 11.2.

Las constantes públicas estáticas de la clase *NCLEvent* son:

- *static* int PRESENTATION_START
    - Máscara de evento para eventos de presentación (tipo *presentation*) cuya acción (campo *action*) fue el inicio de la reproducción (*start*) de un áncora definida para el nudo de media (elemento *<media>*) que incluye el Xlet Ginga-J.
- static int PRESENTATION_STOP
    - Máscara de evento para eventos de presentación (tipo presentation) cuya acción (campo action) fue la terminación de la reproducción (stop) de un áncora definida para el nudo de media (elemento <media>) que incluye el Xlet Ginga-J.
- *static* int PRESENTATION_ABORT
    - Máscara de evento para eventos de presentación (tipo *presentation*) cuya acción (campo *action*) fue abortar la reproducción (*abort*) de un áncora definida para el nudo de media (elemento *<media>*) que incluye el Xlet Ginga-J.
- *static* int PRESENTATION_PAUSE
    - Máscara de evento para eventos de presentación (tipo *presentation*) cuya acción (campo *action*) fue la pausa de la reproducción (*pause*) de un áncora definida para el nudo de media (elemento *<media>*) que incluye el Xlet Ginga-J.
- *static* int PRESENTATION_RESUME
    - Máscara de evento para eventos de presentación (tipo *presentation*) cuya acción (campo *action*) fue la retomada de la reproducción (*resume*) de un áncora definida para el nudo de media (elemento *<media>*) que incluye el Xlet Ginga-J.
- *static* int ATTRIBUTION_SET
    - Máscara de evento para eventos de atribución (tipo *attribution*) cuya acción (campo *action*) fue la definición (*set*) de un parámetro definido para el nudo de media (elemento *<media>*) que incluye el Xlet Ginga-J.
- protected int id
    - Identificación del evento.

Los métodos públicos de la clase *NCLEvent* son:

- NCLEvent(Object source, int id, String value)
    - Método constructor para NCLEvent, que recibe como parámetro una referencia (source) del objeto que originó el evento, un entero (id) que identifica el evento y un identificador (value) del nudo o áncora relacionado al evento.
- int getID()
    - Retorna el tipo de evento.
- String getValue()
    - Retorna el identificador del nudo o áncora relacionado al evento.

D.2.2.3 Interfaz *NCLEventListener*

La interfaz *Listener* debe ser implementada por quien desea recibir notificación de eventos distribuidos por el formateador NCL manejando un documento NCL que incluye un Xlet Ginga-J, siendo objetos que son elementos de la clase *NCLEvent*. Extiende la interfaz java.util.EventListener

La aplicación que desee monitorear los eventos generados por el formateador NCL debe implementar esta interfaz.

El método público de la interfaz *NCLEventListener* es:

- void NCLPlayerEventDispatched(NCLEvent event)
    - Método ejecutado cuando un evento NCLEvent es generado por el formateador NCL.

Anexo E
(normativo)

Especificación API de soporte a planos gráficos – Paquete *br.org.sbtvd.ui*

E.1 Clase *ColorCoding*

La clase *ColorCoding* abriga constantes para enumerar los diferentes modelos de codificación posibles para cada plano. Los posibles valores corresponden a aquellos retornados en com.sun.dtv.ui.Plane. getColorCodingModel().

Las constantes públicas estáticas presentes en esta clase son:

- *public static* final int ARGB8888 = 1
    - indica que el modelo de colores en el plano es ARGB8888
- *public static final int YUV442 = 2*
    - indica que el modelo de colores en el plano es YUV442.
- *public static* final int YUV444 = 3
    - indica que el modelo de colores en el plano es YUV444.
- *public static* final int ONE_BPP = 4
    - indica que el modelo de colores en el plano es un bit por pixel.

E.2 Clase *StillPicture*

La clase *StillPicture* extiende la clase com.sun.dtv.lwuit.Component. Es el medio por el cual imágenes JPEG son adicionadas al plano de imágenes estáticas. Esta clase se debe declarar con el modificador final.

Los constructores públicos son los siguientes:

- *StillPicture(String path)*
    - Construye un objeto StillPicture. El camino pasado en el parámetro path debe corresponder a la localización de una imagen JPEG en el sistema de archivos de la aplicación.

Las instancias de esta clase no soportan las funcionalidades de enfoque ni de animación.

Los siguientes métodos heredados del com.sun.dtv.lwuit.Component no pueden ser invocados directamente por las aplicaciones:

- *void paint(Graphics g)*

- *void paintBackgrounds(Graphics g)*
- *void paintComponent(Graphics g)*
- *void paintComponent(Graphics g, boolean background)*

E.3 Clase *SwitchArea*

La clase *SwitchArea* extiende la clase com.sun.dtv.lwuit.Component. Esta clase se debe declarar con el modificador final.

Es un componente que define un área rectangular del plano de selección video/imagen. Cada área rectangular adicionada por medio del método com.sun.dtv.ui.DTVContainer#addComponent() corresponde a un área en que el plano de imágenes estáticas aparecerá sobre el plano de video o viceversa, dependiendo del color del estilo (*com.sun.dtv.lwuit.plaf.Style*) del componente.

Las instancias de esta clase no soportan las funcionalidades de enfoque ni de animación.

Los siguientes métodos heredados del com.sun.dtv.lwuit.Component no deben ser invocados directamente por las aplicaciones:

- void paint(Graphics g)
- void paintBackgrounds(Graphics g)
- void paintComponent(Graphics g)
- void paintComponent(Graphics g, boolean background)

Por medio del com.sun.dtv.lwuit.plaf.Style asociado a cada componente se puede definir si el video se exhibirá por encima del Still Picture Plane o viceversa. Las instancias de com.sun.dtv.lwuit.plaf.Style asociadas a este componente solamente pueden contener colores sólidos (java.awt.Color). El color negro, java.awt.Color. BLACK, representa que el video se debe exhibir por encima del Still Picture Plane. Con el uso de cualquier otro color, el contenido del Still Picture Plane se exhibirá por delante del video.

Bibliografía

[1] *SOUZA FILHO, Guido Lemos de; LEITE, Luiz Eduardo Cunha; BATISTA, Carlos Eduardo Coelho Freire. Ginga-J: The Procedural Middleware for the Brazilian Digital TV System. In: _____ Journal of the Brazilian Computer Society. No. 4, Vol. 13. p.47-56. ISSN: 0104-6500. Porto Alegre, RS, 2007.*

[2] *SOARES, Luiz Fernando Gomes; RODRIGUES, Rogério Ferreira; MORENO, Márcio Ferreira. Ginga-NCL: the Declarative Environment of the Brazilian Digital TV System. In: _____ Journal of the Brazilian Computer Society. No. 4, Vol. 13. p.37-46. ISSN: 0104-6500. Porto Alegre, RS, 2007.*

[3] *Sun Microsystems, Java Digital Television (DTV) API:2008, < http://java.sun.com/javame/technology/javatv/index.jsp>*

[4] *Sun Microsystems, Java TV API:2007, <http://java.sun.com/products/javatv/>*

[5] *Sun Microsystems, Java Media Framework API (JMF), <http://java.sun.com/products/java-media/jmf/index.jsp>*